Copy No. _____

Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Using Software Agents to Control the Behaviour of Simulated Entities

Tania E. Randall

Defence R&D Canada – Atlantic

Technical Memorandum
DRDC Atlantic TM 2007-342
December 2007

Canada

This page intentionally left blank.

Using Software Agents to Control the Behaviour of Simulated Entities

Tania E. Randall

Defence R&D Canada – Atlantic

Technical Memorandum
DRDC Atlantic TM 2007-342
December 2007

Principal Author

Original signed by Tania E. Randall

Tania E. Randall

Approved by

Original signed by J. S. Kennedy

J. S. Kennedy

Head, Maritime Information and Combat Systems

Approved for release by

Original signed by J. L. Kennedy

J. L. Kennedy

DRP Chair

The information contained herein has been derived and determined through best practices and adherence to the highest levels of ethical, scientific, and engineering investigative principles. The reported results, their interpretation, and any opinions expressed therein, remain those of the authors and do not represent, or otherwise reflect, any official opinion of position of DND or the Government of Canada.

Abstract

The Virtual Combat Systems (VCS) group at Defence R&D Canada – Atlantic (DRDC Atlantic) has been developing a virtual maritime environment for command and control (C2) level, human-in-the-loop experimentation using the High Level Architecture (HLA)-based Virtual Maritime Systems Architecture (VMSA). For typical experiments using VMSA, human operators have been used to control the friendly (blue) force entities during an experiment while the opposing (red) force entities have been controlled by the simulation. Most recently, the red force entities have been modelled within the Joint Semi-Automated Forces (JSAF) environment. While the sensor and weapon models inside of JSAF are believed to be of a good fidelity, the ability to describe an entity's behaviour and decision making is lacking. In addition, JSAF scenarios are tedious to develop and require human input at run-time, creating problems in terms of scenario repeatability.

In this paper, the use of agent-based technology is investigated as a method for improving the definition, autonomy and realism of red force entities for use in VMSA simulations. The JACK Intelligent Agent software was selected for developing behaviours and decision making capabilities for simulation-controlled entities. In the application discussed in this document, JACK is used to decide when an entity should move, share information with another entity, or fire its weapons, while VMSA is used to model the physical aspects of the entities, such as motion, sensors and weapons.

A scenario involving a group of fast inshore attack craft (FIACs) claiming control over a narrow strait is modelled. The paper focuses on the process of taking a conceptual description of this scenario and mapping it into agent constructs (events, goals, plans, and beliefs) that are meaningful to the JACK software. It concludes that agent technology *can* be used to effectively control computer generated forces in a VMSA simulation and discusses the benefits of this approach.

Résumé

Le Groupe des systèmes de combat virtuel (SCV) de Recherche et développement pour la défense Canada – Atlantique (RDDC Atlantique) a développé un environnement maritime virtuel destiné à l'expérimentation au niveau commandement et contrôle (C2) avec la participation de personnes réelles, en faisant appel à l'architecture VMSA (architecture de systèmes virtuels maritimes), elle-même fondée sur HLA (*High Level Architecture)*. Dans les expériences-types utilisant VMSA, on a eu recours à des opérateurs humains afin de contrôler les entités de la force amie (bleue), alors que les forces opposées (rouges) étaient contrôlées par la simulation. Plus récemment, les entités des forces rouges ont été modélisées au sein de l'environnement JSAF (forces interalliées semi-automatisées). Bien que les modèles des capteurs et des armes intégrés à JSAF soient considérés comme ayant un bon niveau de fidélité, ils ne comportent pas de description du comportement et des prises de décision des entités. De plus, l'élaboration de scénarios en JSAF est fastidieuse et elle exige la participation de personnes au moment de l'exécution, ce qui rend problématique la répétabilité des scénarios.

Ce rapport examine l'usage de la technologie fondée sur les agents comme méthode d'amélioration de la définition, de l'autonomie et du réalisme des entités de la force rouge dans les simulations VMSA. Le logiciel JACK Intelligent Agent a été sélectionné afin d'élaborer des comportements et capacités de prise de décision s'appliquant à des entités contrôlées par des simulations. Dans l'application présentée dans le présent document, JACK est utilisé afin de décider du moment où les entités se déplacent, partagent des informations avec d'autres entités ou font usage de leurs armes, alors que VMSA est utilisé afin de modéliser les aspects matériels des entités comme le mouvement, les capteurs et les armes.

Un scénario mettant en jeu un groupe d'embarcations d'attaque côtière rapides (FIAC) tentant de prendre le contrôle d'un passage étroit est modélisé. Ce rapport porte principalement sur le processus qui consiste à utiliser une description conceptuelle de ce scénario et à le faire correspondre à des constructions d'agents (événements, objectifs, plans et croyances) significatifs pour le logiciel JACK. Ce rapport conclut que la technologie des agents *peut* être utilisée afin de contrôler efficacement des forces générées par ordinateur dans une simulation VMSA et il discute des avantages de cette approche.

Executive summary

Using Software Agents to Control the Behaviour of Simulated Entities

Randall, T.E.; DRDC Atlantic TM 2007-342; Defence R&D Canada – Atlantic; December 2007.

Background: The Virtual Combat Systems (VCS) group at Defence R&D Canada – Atlantic (DRDC Atlantic) has been developing a virtual maritime environment for command and control (C2) level, human-in-the-loop experimentation using the High Level Architecture (HLA)-based Virtual Maritime Systems Architecture (VMSA). For typical experiments using VMSA, human operators have been used to control the friendly (blue) force entities during an experiment while the opposing (red) force entities have been controlled by the simulation. Most recently, the red force entities have been modelled within the Joint Semi-Automated Forces (JSAF) environment. While the sensor and weapon models inside of JSAF are believed to be of a good fidelity, the ability to describe an entity's behaviour and decision making is lacking. In addition, JSAF scenarios are tedious to develop and require human input at run-time, creating problems in terms of scenario repeatability. This paper explores the use of agent-based technology as a method for improving the definition, autonomy and realism of red force entities for use in VMSA simulations.

Results: The JACK Intelligent Agent software was chosen for developing the behaviours and decision making capabilities for simulation-controlled VMSA entities. This paper looks at using JACK agents to control a group of fast inshore attack craft (FIACs) claiming control over a narrow strait. Agents are used to decide when a FIAC should move, share information with another entity, or fire its weapons, while VMSA is used to model the physical aspects of the entities, such as motion, sensors and weapons. The paper focuses on the process of taking a conceptual description of the scenario and mapping it into agent constructs (events, goals, plans, and beliefs) such that it can be implemented in the JACK software. This process was not as straight forward as anticipated and required a fair bit of trial and error within the JACK software to figure out details that could not be found in the software documentation or other literature resources. Typically papers in the literature do not focus on the details of how JACK was used, rather on general ideas that *could* be implemented with JACK, or the results of systems that were actually developed. While ultimately it is those 'results' that are of value, understanding how those results were obtained is equally important since it allows the reader to make an informed decision about how much weight to attribute to them. It is also hoped that this documentation will be of use to future JACK users. This paper highlights the process of creating the design that was implemented in JACK, and demonstrates how events, goals, plans and beliefs have been used to create entity behaviours for VMSA simulations. It concludes that agent technology *can* be used to effectively control computer generated forces and discusses the benefits of this approach: repeatable scenarios, more varied and realistic red force behaviours, and a longer term potential for faster scenario development.

Significance: This paper demonstrates the process of using agent-based intelligence to control the enemy forces for VMSA simulations. Having red forces which exhibit realistic behaviours and decision making capabilities is critical to the success of virtual environment-based experiments as

well as the credibility attributed to it by military operators participating in the experimentation program. By using agents to control entities, the entities act autonomously, react to their environment, and make proactive decisions to act in ways that will help them achieve their goals, much as humans do. Their decisions are context-sensitive, so the response to an event may vary depending on the agent's current situation, making the entity seem more intelligent. An agent will make the same decision each time it is presented with a given (identical) situation, so the scenarios are repeatable. Agents are a plausible alternative to JSAF for modelling decisions and behaviours for non-human controlled entities in the VMSA simulated environment.

Future plans: As JACK is used to develop different or more complicated scenarios, it is likely that new agent types and corresponding events, plans and beliefsets will need to be created and the software bridge may need to be extended to pass more types of information between the agents and the VMSA simulation. However, as entities and behaviours are built up, the development of new scenarios will become easier. The longer term goal involves the development of tools and processes to make military scenario development for simulated environments less time consuming by allowing the developer to specify broader details (for example, a fishing village rather than individual fishing boats), and for the resulting entity behaviours to better reflect those of the real world.

Sommaire

Using Software Agents to Control the Behaviour of Simulated Entities

Randall, T. E.; DRDC Atlantic TM 2007-342; R &D pour la défense Canada – Atlantique; décembre 2007.

Contexte : Le Groupe des systèmes de combat virtuel (SCV) de Recherche et développement pour la défense Canada – Atlantique (RDDC Atlantique) a développé un environnement maritime virtuel destiné à l'expérimentation au niveau commandement et contrôle (C2) avec la participation de personnes réelles, en faisant appel à l'architecture VMSA (architecture de systèmes virtuels maritimes), elle-même fondée sur l'architecture HLA (*High Level Architecture*). Dans les expériences-types utilisant VMSA, on a eu recours à des opérateurs humains afin de contrôler les entités de la force amie (bleue) alors que les forces opposées (rouges) étaient contrôlées par la simulation. Plus récemment, les entités des forces rouges ont été modélisées au sein de l'environnement JSAF (forces interalliées semi-automatisées). Bien que les modèles des capteurs et des armes intégrés à JSAF soient considérés comme ayant un bon niveau de fidélité, ils ne comportent pas de description du comportement et des prises de décision des entités. De plus, l'élaboration de scénarios en JSAF est fastidieuse et elle exige la participation de personnes au moment de l'exécution, ce qui rend problématique la répétabilité des scénarios. Ce rapport examine l'usage de la technologie fondée sur les agents comme méthode d'amélioration de la définition, de l'autonomie et du réalisme des entités de la force rouge dans les simulations VMSA.

Résultats : Le logiciel JACK Intelligent Agent a été sélectionné afin d'élaborer des comportements et capacités de prise de décision s'appliquant à des entités contrôlées par des simulations. Le présent rapport examine l'utilisation d'agents JACK pour contrôler un groupe d'embarcations d'attaque côtière rapides (FIAC) tentant de prendre le contrôle d'un passage étroit. Les agents sont utilisés afin de décider du moment où une FIAC doit se déplacer, partager de l'information avec une autre entité ou utiliser ses armes, alors que VMSA est utilisé afin de modéliser les aspects matériels des entités comme le mouvement, les capteurs et les armes. Le rapport porte principalement sur le processus qui consiste à utiliser une description conceptuelle d'un scénario et à le faire correspondre à des constructions d'agents (événements, objectifs, plans et croyances) en vue de leur intégration dans JACK. Ce processus n'a pas été aussi simple que prévu et il a été nécessaire de faire appel au processus d'essais et d'erreurs dans JACK afin d'en déduire des détails qui ne figuraient pas dans la documentation du logiciel ou d'autres sources documentaires. Habituellement la documentation ne détaille pas l'utilisation de JACK, elle décrit plutôt des idées générales qui *pourraient* être mises en oeuvre au moyen de JACK ou encore les résultats obtenus avec des systèmes qui ont été développés. Bien que ce soit ces « résultats » qui comptent finalement, il est aussi important d'expliquer comment ils ont été obtenus car cela permet aux lecteurs de prendre des décisions informées au sujet du poids qu'on peut leur accorder. Nous espérons de plus que cette documentation sera utile pour les utilisateurs futurs de JACK. Ce rapport décrit le processus de conception appliqué à JACK et il montre comment on a utilisé des événements, objectifs, plans et croyances afin de créer des comportements d'entité en vue des simulations VMSA. Ce rapport conclut que la technologie des agents peut être utilisée afin de contrôler efficacement des forces générées par ordinateur il discute des avantages de cette

approche : scénarios répétables, comportement de la force rouge plus varié et réaliste, ainsi que, à plus long terme, potentiel de développement plus rapide des scénarios.

Portée : Ce rapport fait la démonstration du processus d'utilisation de l'intelligence fondée sur des agents afin de contrôler les forces ennemies dans des simulations VMSA. Des forces rouges se comportant de manière réaliste et disposant de capacités de décision réalistes sont essentielles à la réalisation d'expériences basées sur un environnement virtuel. Elles sont aussi requises afin que les opérateurs militaires participant à un tel programme d'expérimentation puissent leur accorder de la crédibilité. En utilisant des agents pour contrôler des entités, ces dernières agissent de manière autonome, réagissent à leur environnement et prennent des décisions proactives afin d'agir de manière à atteindre leurs objectifs, de façon assez semblable aux humains. Leurs décisions dépendent du contexte et leur réaction à un événement changera selon la situation actuelle de l'agent, ce qui fait paraître plus intelligente l'entité. Par ailleurs, un agent prend la même décision à chaque fois qu'il se trouve dans une situation donnée (identique), ce qui permet que les scénarios soient répétables. Les agents constituent une solution plausible de remplacement de JSAF pour effectuer la modélisation des décisions et des comportements d'entités non contrôlées par des humains dans l'environnement simulé en VMSA.

Plans futurs : Si JACK est utilisé pour élaborer des scénarios différents ou plus complexes, il est probable que de nouveaux types d'agents et d'événements, plans et ensembles de croyances correspondants devront être produits. Il pourra aussi être nécessaire d'étendre le pontage logiciel afin de transmettre un plus grand nombre de types d'information entre agents et simulations VMSA. Par ailleurs, l'élaboration de nouveaux comportements et entités facilitera l'élaboration de nouveaux scénarios. À plus long terme, l'objectif sera de développer des outils et processus qui réduiront le temps requis pour le développement de scénarios militaires s'appliquant à des environnements simulés en permettant au développeur de définir des éléments de manière plus générale (par exemple, en décrivant un village de pêcheurs au lieu d'avoir à décrire les navires de pêche individuels), et pour que le comportement des entités résultantes corresponde mieux à celui du monde réel.

Table of contents

List of figures

List of tables

Acknowledgements

The author would like to acknowledge the support provided by Briand Gaudet of SG Software Solutions throughout the design process. He was responsible for developing the JACK-VMSA software bridge and was a great help in understanding the requirements and capabilities of the JACK software.

This page intentionally left blank.

1 Background

1.1 The VCS Group and CGFs

The Virtual Combat Systems (VCS) group at Defence R & D Canada – Atlantic (DRDC Atlantic) has been developing a virtual maritime environment for command and control (C2) level, human-in-the-loop (HIL) experimentation, using the High Level Architecture (HLA)-based Virtual Maritime Systems Architecture (VMSA) [1]. In one way or another, these experiments have dealt with the operator's ability to detect and identify entities, particularly the hostile ones. An entity's behaviour is perhaps the greatest indicator of its intent, prior to an attack. While the type of entity as well as observation of weapons can indicate that the entity has an ability to inflict damage, it is the entity's behaviour that portrays its intent. Behaviour modelling is also important to simulation credibility. A vessel travelling at unrealistic or impossible speeds, for example, brings the credibility of the entire system into question.

Still, the modelling of entity behaviour in VMSA simulations to date has been less than optimal. In 2003, when the VCS Group was just ramping up, these computer-controlled entities (known as computer generated forces (CGFs)) were no more than entities of a specified type placed on a game board prior to run-time, with a series of waypoints to follow and speeds to maintain. Regardless of what happened around them (for example, a human-controlled entity firing at it), the entity performed as initially planned.

In more recent experiments, such as the Virtual Battle Experiment–Echo (VBE-E) [2], Joint Semi-Automated Forces (JSAF) was used to control both neutral and red force entities. While it took significant effort to get JSAF integrated with a VMSA federation, the use of JSAF was a step in the right direction. Entity behaviour became more sophisticated, with the ability to have an entity follow a path, travel to a point, pause, etc.. When given the proper permissions, these entities would also fire upon opposing force entities when in range. The red (and neutral) force game controller could also change the actions of these entities on the fly (i.e., at run-time) by modifying destination points, path plans, speeds, etc.. Setting up a JSAF scenario, however, took considerable time and patience as JSAF is not a commercial product and is neither user friendly nor intuitive. Since most VBE-E scenarios involved approximately one hundred entities and approximately fifteen scenarios were required, spending a minute or so specifying the details of each entity became quite time consuming. Also, while the option for JSAF operators to change entity actions on the fly may be a useful feature for training exercises, it has the potential to sabotage experimental results. To achieve statistical confidence in results, the experimenters must be able to control the factors that change from one run to the next, in order to be able to attribute changes in the results of a run to a specific cause.

There are a few main considerations when selecting appropriate software for scenario development and execution for experimentation: how realistic the scenario appears at runtime, the repeatability of the scenario, and the amount of time and effort that it takes to design and set up a scenario. Neither of the existing scenario development methods excels at all of these. This issue spawned an investigation into the use of agents for scenario development and entity behaviour modelling for experimentation. A review of agent-based technology can be found in [3]. Following this review, JACK Intelligent Agents Software ([4],[5]) was purchased for agent

development. Using agents, behaviours for entity types can be defined. Agents can make context-sensitive decisions about what actions to take, so their decisions can appear more realistic. They are also autonomous, so no operator input is required at run-time. Given the same actions of any human-controlled entities, the scenario will play out the same way a second time, so the scenarios are repeatable. Once agents exist for each entity type, scenario development is reduced to defining the type of agent controlling each entity and providing some initialization data. In the longer term, the hope is to eliminate the requirement to define specific positions for each entity over time. Instead, the user would define types of areas containing certain types of entities and let the simulation work out the finer details.

A project was initiated to integrate VMSA with JACK Intelligent Agents and demonstrate the feasibility of using agents to control the red forces. A swarm attack scenario was chosen for this demonstration since it is relevant to current VCS group projects (e.g., force protection) and can be modified to meet the specific needs of these projects without too much effort. VMSA and JACK were integrated such that the agents control all of the decision making (e.g., where to go, when to fire) for the enemy forces, while VMSA handles all physical modelling such as sensors, weapons, and motion. The decisions for friendly forces continue to be controlled through human input to the VMSA simulation.

Considerable effort was required to take a conceptual scenario and map it into the constructs of the JACK modelling environment. In JACK, all agents are defined in terms of events, goal events, plans, and beliefsets. This paper focuses on the transition of a conceptual scenario into these JACK constructs. Documentation on the development of the software bridge to pass messages back and forth between VMSA and JACK can be found in [6]. The paper concludes with a discussion on how, for this scenario in particular, the use of agents to control the red force entities compares to the use of former methods.

1.2 Agents

While it is hard to find an agreed upon definition of a software agent, it is generally accepted that an agent is a software entity that is:

- autonomous - able to act independently of any human input and can make decisions for itself;
- reactive - able to react to changes in its environment or events that occur (e.g., it might run away when fired at); and
- proactive - able to act not only in response to things that occur, but also act based on an inherent desire for the world to be a certain way (e.g., to destroy another entity).

1.3 JACK Intelligent Agents Development Environment

JACK is “an environment for building, running and integrating commercial-grade multi-agent systems using a component-based approach” [4]. JACK is based on JAVA and extends JAVA with methods for agents, events, goal events, plans, and beliefsets. The designer must create each of these constructs and build the required functionality into them using JAVA and the JACK Agent Language (JAL). The JACK engine then knows how these agent-oriented concepts relate

to one another and is able to progress the simulation as follows: an event occurs and JACK looks for a plan that the agent has that can handle that event. If no plan exists, the agent does nothing. If plans do exist which handle that event, JACK checks to see if the plan(s) is (are) both relevant and applicable, given the current situation. If more than one plan is relevant and applicable, further reasoning can be used to decide which of those plans to select. Once a plan is selected, JACK executes that plan. Within a plan, more events can be posted, in which case a subtask is created to handle that new event, and after the subtask is complete, execution will return to the original plan. Other events may occur externally to the currently executing plan as well. These events can be messages from other agents or based on new information from one of the agent's sensors. A plan to handle that event may then also begin executing in parallel with the other plan. JACK handles the sharing of processing between all the current tasks, not just for a single agent, but for all the agents currently executing one or more plans.

1.4 'Bridging' JACK and VMSA

Figure 1 *p*rovides an overview of how JACK and VMSA have been used together to model the red forces. JACK is used for modelling the intelligence of the red entities, which for this initial project involves deciding when an entity should move and when it should fire its weapon. VMSA handles the modelling of all physical components: motion, visuals, damage and weapons for the red entities as well as the blue entities. The intelligence for the opposing blue force entities is provided by a human operator through a command and control (C2) interface. VMSA passes information to the agents about the positions of each red entity, what each entity can see, the damage level of each entity that the red force entity can see, and the location of detonations/bullet landings. The passing of JACK agent data to VMSA and VMSA data to JACK is handled by a software bridge. More information on the JACK-VMSA bridge can be found in [6]

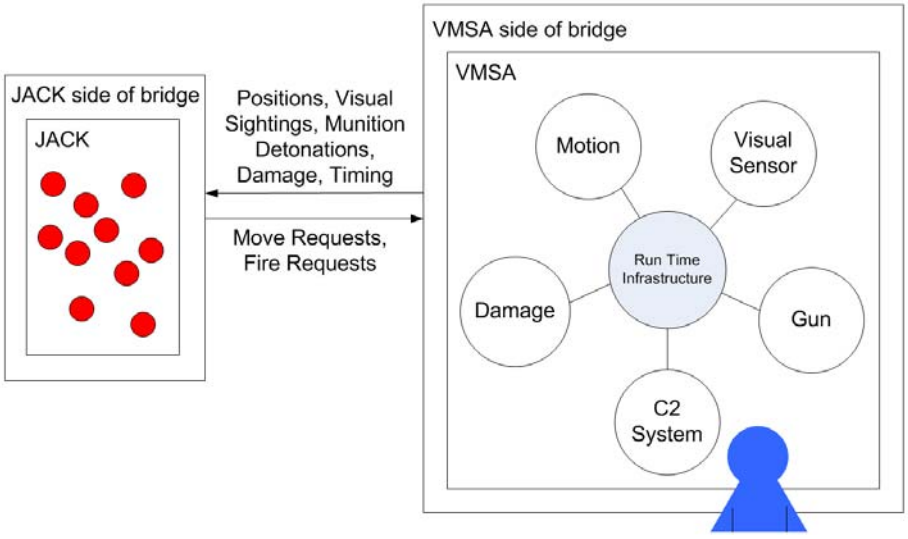

Figure 1: Responsibilities of JACK and VMSA

This paper addresses the design of the JACK agents and their interaction with the physical models in VMSA. The agents can be thought of as the humans controlling the red entities. The requests

to VMSA for movement or weapon fire correspond to a human steering the boat or launching a weapon. Neither the VMSA components nor the JACK-VMSA bridge will be discussed in detail in this paper. They will only be referred to the extent that they influence the design of the JACK agents.

1.5 A Design Methodology

The Prometheus agent design methodology ([7], [8]) supports the development of intelligent agents which use goals, beliefs, plans, and events. It consists of three phases: system specification, architectural design, and a detailed design phase. At the beginning of this project, it was thought that the Prometheus design methodology would be the best way to define the problem at hand in terms of an agent system. However, it turned out that the documents available online ([7], [8]) which summarize this methodology, do not provide enough information (e.g., detailed examples) to effectively work through it. These papers summarize a book, [9], which has the necessary detail, but which was unavailable within the project timelines. More importantly though, formal agent methodologies attempt to describe the development process in a general manner so that the resulting design could be implemented using one of various software tools, rather than a specific one, such as JACK. Prometheus is, however, focussed on systems that use the Beliefs-Desires-Intentions (BDI)-paradigm as JACK does, and is more architecture-specific than some other methodologies. The designers of Prometheus argue that it is more important that a methodology be useful than general. In this document, that argument is taken a step further. At the onset of the project, it was already known that JACK would be the software system used for implementation. So, instead of thinking about agents in a generic sense, the knowledge of JACK's requirements was used to guide the project design phase. It is possible that a formal methodology such as Prometheus could be preferable for certain complicated systems, but for this project a more software-specific approach was taken. It can be summarized as follows:

1. Describe the general scenario that will be modelled without consideration for *how* it will be modelled. This is the conceptual model; a simplification of a real-world scenario.
2. From the general scenario, identify the agent types that are required. Agents of a given type have access to the same plan, event and beliefset types to determine their actions. In terms of agents for red force modelling, it is necessary to decide if a single agent will control multiple red force entities, a single red force entity, or only a functional component of a red force entity (e.g., the motion, or, the weapons).
3. Describe the general scenario again with more technical detail, paying attention to how each component of the scenario will be achieved in the modelling world (e.g., data needed and recorded and shared). At this point, the description is still independent of JACK's requirements.
4. Look at the requirements of the JACK development environment and consider how the existing scenario description can be turned into something meaningful to JACK. This can be handled through a series of JACK-specific steps. For each agent type, identify:
    a. the events and goal events that will drive the system,
    b. the plans that can be used to handle the events, and when they are relevant and applicable,

c. the beliefset data that the agent will need when selecting plans to handle events,

d. the hierarchy of goal events (e.g., which goal overrides or suspends another goal)

e. which agent types use which events, plans and beliefsets.

5. Use the JACK Design tool to create diagrams showing the relationships amongst the events, plans, beliefsets and agents described above. Describe the scenario in terms of these JACK constructs.

After these five steps are complete, the design aspect of the agent system will be finished and the appropriate code can be added to the plans, events and beliefsets.

The next sections discuss the application of this ‘methodology’ to a simple swarm attack scenario that has been implemented in JACK.

2 Scenario Description

This section describes the scenario that has been modelled using the JACK agent software. For now, the scenario is described without consideration for the software that is used for implementation. Section 2.1 provides a brief background for the current situation as well as an initial description of the scenario. Section 2.2 identifies the types of agents that are required to model the scenario. Section 2.3 outlines the scenario in more detail, focussing on the data that the agents need to use and record.

2.1 Scenario Background and Initial Description

2.1.1 Background

Two nations bordering the opposite sides of a narrow east-west strait have been in ongoing conflict with each other. It is widely known that any vessel passing through this strait could be in danger of attack by insurgents from one of these nations who want to show that they are in control of the strait. It is also known that if a vessel concedes control to these insurgents by paying them a fee, they will be permitted a safe passage. If arrangements with insurgents are not made and an attempt to pass through the strait still occurs, consequences are almost certain. The navy of a neutral, peaceful country has been brought in to show these insurgents that they do not have the control that they think they do. Two frigates are stationed within the strait to monitor the daily activities, provide a feeling of safety to incoming vessels, and halt insurgent attacks on innocent vessels if it becomes necessary. The insurgents feel very strongly about their cause, however, and have no intentions of surrendering their control. While they will not directly attack a naval vessel, they refuse to have them interrupt their mission. They will try to remain hidden amongst other neutral vessels in the strait until such time that an attack becomes necessary.

2.1.2 Initial Scenario Description

Red attack forces in small fishing craft are dispersed throughout the eastern portion of the strait, attempting to blend in with the other neutral, small vessel traffic, such as fishing boats and pleasure craft (including neutral small fishing craft), which have been permitted by the insurgents to continue to carry out normal activities. A red spotter remains at the eastern end of the strait, monitoring all incoming traffic using visuals (eyes/binoculars) only. It is located in a narrow portion of the strait, where it will be possible to fully identify all vessels traversing the strait once they get close enough. Thus, the spotter's job is very simple: stay put and watch for intruders; if an intruder is identified, inform the others. The informed entities then head for one of two central gathering areas (whichever is closest), located on opposite sides of the strait, travelling at relatively slow speeds, in an attempt to not draw attention to themselves. It is from these areas that the attack will begin. Once the intruder passes a pre-defined location (the 'attack line'), all of the attackers head towards it at full speed. The attackers are armed with rocket propelled grenades (RPGs) with a known maximum effective range. They will attempt to keep their weapons hidden until they are within useable range. Once within that range, each attacker will begin launching grenades at the intruder until it meets some pre-determined condition for ending

the attack. At any point throughout the scenario, if a detonation occurs near an attacker or spotter, it will react by temporarily altering its course away from the detonation.

A visual representation of the scenario is provided in Figure 2. The 'Not Modelled' portion of the diagram will be discussed in Section 2.3.

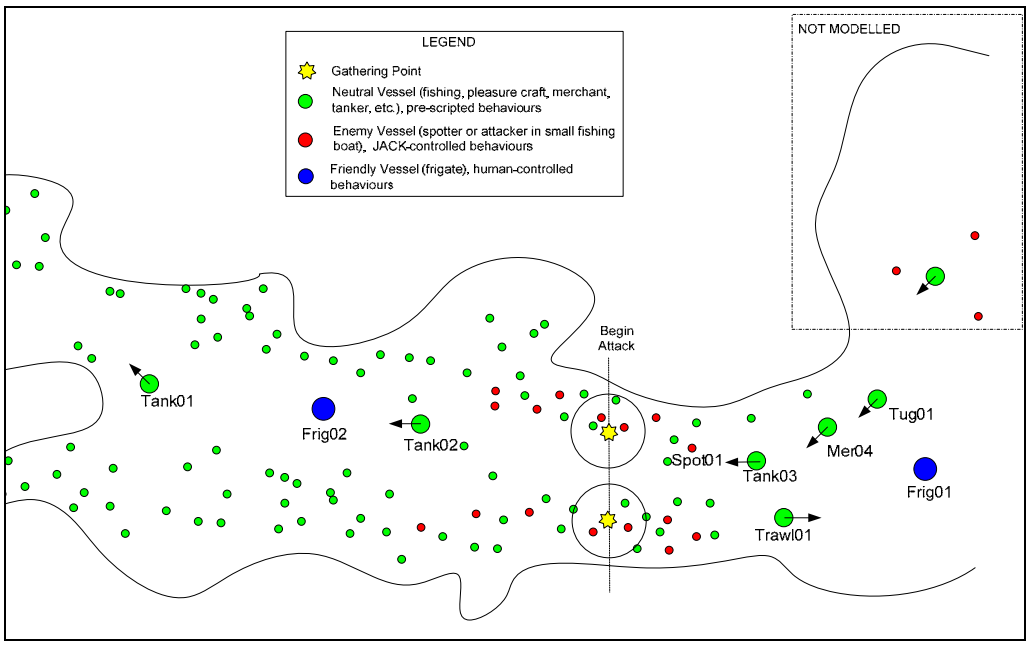

Figure 2: Visual Representation of the Scenario

2.2 Agent Types

For the initial implementation of this scenario, two types of agents have been used: spotter agents and attacker agents. There is one agent per entity. The spotter agent represents a small fishing craft acting in a spotter role, while an attacker agent represents a small fishing craft acting in an attacker role. While the spotter and attackers share the same overall goal to 'defend their strait', the spotter and attackers have different individual goals and plans to achieve them, so different agent types are required. It is true, however, that the agent types have some common capabilities.

2.3 Detailed Scenario Description

This section describes the important elements of the scenario in more detail, paying close attention to the initial data that the agents will need to make their decisions and the data that the agents will receive and record from their environment.

Rather than specifying numerical data in the tables that follow, italicized text strings will be used as placeholders. For example, instead of '10 metres', '*range1*' could be used. Non-italicized text strings, for example, 'Merchant', are meant to represent actual values that could appear in the tables.

As discussed previously, the insurgents are claiming that any large vessel that wishes to pass through the strait must either pay a fee to them or face the consequences. It is assumed that there are some additional members of the red team (the 'negotiators') in the eastern mouth of the strait, managing these transactions. If a vessel refuses to pay, it will be given a warning by the negotiators, but will receive no further consequences at that time. The negotiators will then inform the in-strait spotter of the approaching intruder. The job of this spotter is then to re-locate any unapproved vessels as specified by the negotiators. The negotiators themselves will not be modelled (as indicated in Figure 2). The simulation will begin with the agents already knowing which vessels are the unwelcomed ones, i.e., the intruders.

Table 1: Intruders

Intruders		
Name	**Type**	**ID**
SeaSurface.NonMilitary…Merchant...Mer04	Merchant	Mer04
SeaSurface.Military.Warship.Frigate….Frig01	Frigate	Frig01
SeaSurface.Military.Warship.Frigate….Frig02	Frigate	Frig02

The Name field holds the VMSA truth name of the entity. Both the Type and ID could be extracted from the Name string at run-time, however, they have been included in separate fields as well to facilitate any searches that involve looking for the type or ID of an entity.

The red forces will treat the non-frigate intruders differently than they will treat any commercial intruders. The red forces know that the frigates are equipped with substantial fire power and do not wish to challenge them. It is the commercial vessels that are attempting to pass through the strait without paying their dues that will become the focus of the red force's disapproval. The frigates have been on location for a few days and are aware that these enemy forces are likely to be present; however, they are unable to differentiate them from the other traffic in the strait as long as they do not act out of character for a truly neutral vessel.

Since the spotter does not want to draw attention to itself and it knows that it will be able to identify any large vessel from its current position in the strait, it does not move far from its initial position unless it becomes threatened by weapon fire. It tries to stay in place and watch incoming traffic.

The spotter's visual detection list may look similar to that shown in Table 2.

Table 2: Detections

Detections					
Name	**Type**	**ID**	**Bearing (degrees)**	**Range (m)**	**Damage (0-4)**
SeaSurface.NonMilitary…Tanker...Tank03	Tanker	Tank03	*bearing1*	*range1*	*damage1*
SeaSurface.NonMilitary…Seadoo...Sea01	Seadoo	?	*bearing2*	*range2*	*damage2*
SeaSurface.NonMilitary…Seadoo...Sea02	Seadoo	?	*bearing3*	*range3*	*damage3*

Note that damage is modelled as part of the visual track. This can be interpreted as damage 'as perceived by the agent's visual sensing'.

While the Name field includes the true identity of the detected entity, this information will not be used by the agent. The Name field is used as the key to the table and prevents the agent from having more than one track on a particular entity. The agent can only use the information in the Type and ID fields for its decision making. In some cases, the agent will know the type, but not the ID. As the entity gets closer to the agent-controlled entity, the rest of the track information will become available. Whenever the spotter obtains an ID on an entity, it checks the intruders list to see if there is a match to anything besides a frigate. If not, nothing happens. If there is a match, the spotter informs each of the attackers (listed in Table 3) of the approaching intruder, triggering the attackers to head for their gathering locations (listed in Table 4). There is a gathering area on both sides of the strait and attackers head for the closest area. The areas are determined prior to run-time, and are defined by a point and a distance from that point (i.e., a circular area).

Table 3: Red Team Names

RedTeamNames
Name
SeaSurface…FishBoat...Attacker01
SeaSurface…FishBoat...Attacker02
: :
SeaSurface…FishBoat...Attacker10

Table 4: Gathering Areas

GatheringAreas			
PointName	**Latitude (degrees)**	**Longitude (degrees)**	**Range (m)**
GatherForAttack_1	*latitude1*	*longitude1*	*range1*
GatherForAttack_2	*latitude2*	*longitude2*	*range2*

Once the attackers reach the attack location, they wait for the target entity (i.e., the non-frigate intruder) to pass the pre-defined 'attack start' line. Within these gathering areas which are located along another narrow portion of the strait, all entities should be able to see (with binoculars) any large entity crossing anywhere along the attack line. So, for this initial demonstration, no track sharing between attackers has been modelled. When the target crosses the line, the attackers head towards the target at full speed.

Each agent should know its own position, course and speed as shown in Table 5. Positional information will also be required for determining how far the entity is from another location (e.g., a gathering point).

Table 5: Self Data

Self				
Name	**Latitude (degrees)**	**Longitude (degrees)**	**Course (degrees)**	**Speed (m/s)**
SeaSurface…FishBoat...Attacker01	*latitude1*	*longitude2*	*course1*	*speed1*

The attack line is defined by two points on either side of the strait (in lat/long decimal degrees). A third point, referred to as the 'test point' will also be included in this table (Table 6). This point is needed for determining which side of the line an entity is on. (Given this point and the position of the entity, the bearing and distance from this point to the entity can be determined. If this distance is less than the distance between the test point and the attack line along the same bearing, then the entity has passed the line.)

The attackers are armed with RPGs which are only effective within a certain range. The attackers will try to keep their RPGs hidden until they are within that range. Once in range, each attacker will begin launching grenades at the target until:

- it runs out of grenades;
- it believes the mission has been accomplished; or
- more than an acceptable number of team members have been destroyed.

The data needed to identify each of these conditions are discussed next.

The number of grenades that each attacker has is fixed at start-up. Weapon range, status, and remaining ammunition are passed from the VMSA simulation to the individual attacker agents. The attacker agents therefore require the data shown in Table 7.

Table 6: Attack Line

LineMarkers			
PointName	**Latitude (degrees)**	**Longitude (degrees)**	**TestPoint**
LineMarker_1	*latitude1*	*longitude1*	False
LineMarker_2	*latitude2*	*longitude2*	False
TestPoint	*latitude3*	*longitude3*	True

Table 7: Weaponry

Weaponry					
WeaponName	**Weapon**	**Range (m)**	**Ammunition**	**Status**	**Ammunition Type**
Launcher_1	RPG	*range1*	*ammunition_count1*	Ready	grenade

For the initial demonstration scenario, the agent will not do anything with the gun status information. However, the gun's status will affect how the gun is operating on the VMSA side (e.g., nothing will be fired while the launcher is being reloaded). The agents simply decide when they want to fire and this desire to fire is limited through VMSA by the capability of the weapon.

Mission accomplishment is based on the total amount of damage applied to the trespasser. The amount of damage required to be considered successful is a preset agent belief (Table 8). Damage levels are enumerated as: 0-None, 1-Slight, 2-Moderate, 3-Severe, 4-Killed. Since the attacker's main goal is to punish the target for entering the strait, and to prevent future attempts, they are not trying to destroy the target, just cause enough damage to 'ruin their day'. Perceived current damage to the intruder must be obtained from the visual detections data.

Table 8: Mission Success

MissionSuccess	
Name	**Damage (0-4)**
SeaSurface.NonMilitary…Merchant...Mer04	*damage1*

The individual agent's tolerance to the destruction of other team members is set at start-up (Table 9). If more than this number of team members is destroyed, the entity will retreat.

Table 9: Team Loss Tolerance

TeamLossTolerance
Limit
tolerance1

When the attackers try to attack the intruder, the human controlled frigates may try to stop them. The frigates are equipped with two types of weapons: multiple .50 calibre machine guns and a 57mm main gun. If 50 cal bullets land near an attacker, the attacker may decide to move away from the landing location (since it may expect future bullets to land close by). It could modify its current course by veering away from that location until it has achieved a seemingly 'safe' distance. It will react similarly to the 57mm shell detonation locations, although a greater escape distance is desired in this case. In the VMSA FOM, the term 'Munition Detonation' is used very loosely to refer to the location that a bullet/shell has landed/detonated. For consistency, this terminology is used in Table 10 and this document as well.

Table 10: Muntion Detonation Responses

MunitionDetonationResponses	
Type	**SafeDistance (m)**
50 cal	*distance1*
57 mm	*distance2*

Whenever a bullet/shell lands/detonates closer than the pre-specified distances, the agent will temporarily modify its course. The red entity is assigned a new, intermediate, destination that is a 'safe' distance from the detonation (as described in Table 10) which will move it away from the detonation, but still move it towards its ultimate destination (e.g., an intruder), as shown in Figure 3. After this intermediate destination has been reached, the agent will once again head towards its original destination.

These reactions are not intended to be optimal; at this point they are simply meant to show that (1) the agent can react in *some way* to weapon fire, and (2) that the agent can differentiate between a 57mm detonation and a 50cal bullet. Munition detonation locations are sent to the agents by VMSA via the JACK-VMSA bridge, but these locations are not be stored by the agents as there is no need for long term knowledge of them.

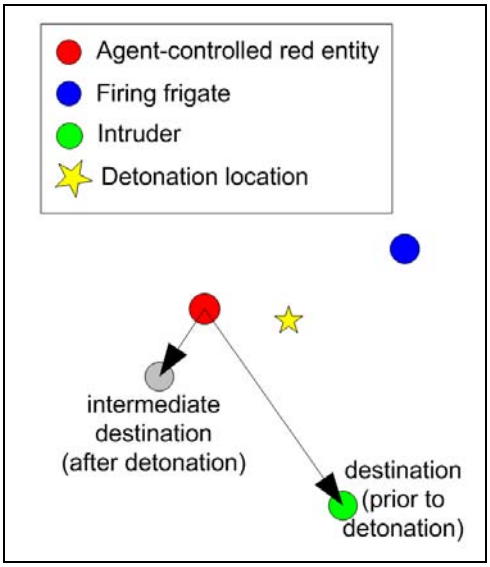

Figure 3: Course Modification to Maintain Safety

3 JACK Events, Plans, Beliefsets, and Agents

3.1 Events

In JACK, all activity is event-driven. If there are no events, none of the agents will do anything. The following is a list and brief description of all of the events that the spotter and/or attacker agents are able to handle.

- *StartSimulation_Event*: Automatically posted by the bridge to all agents when the simulation begins.
- *VesselIDDetermined_Event*(EntityName): Triggered when the ID of a previously unknown vessel becomes available. Sent from spotter to self.
- *IntruderDetected_Event*(EntityName): The spotter has determined the ID for a vessel and it matches that of a vessel in the intruder list. Sent from the spotter to attacker.
- *MoveChoice_Event*(ChoiceType): Triggered when the entity needs to choose between more than one point, entity, or direction (specified by the ChoiceType parameter). Sent from an agent to self.
- *ChoosePoint_Event*: Automatically posted by JACK when there is more than one applicable plan for the *MoveChoice_Event* and there is a need to choose between more than one point.
- *Move_Event*(EntityName, Position, Course, Speed): triggered when the entity has a new destination (either another entity, a collection point, or a direction) or simply a new Speed. Sent from an agent to self.
- *VesselCrossedAttackLine_Event*(EntityName): An entity crosses the attack line. Sent from attacker to self.
- *Attack_Event*(EntityName): Posted to tell the attacker to get within weapon range of the intruder or fire if it is within range. Sent from attacker to self.
- *EndAttack_Event*: Posted when mission has been accomplished (i.e., visual track on target indicates desired amount of damage has been applied to the target), or when the number of team members destroyed exceeds the agent's tolerance, or when the agent runs out of weapons. Sent from attacker to self.
- *MunitionDetonation_Event*(EntityName, MunitionType, Range, Bearing): triggered when munition of type MunitionType from one of the frigates (EntityName) detonates/lands at a bearing of Bearing and range of Range from the agent that receives the message. Sent by bridge to all spotters and attackers that are within a 'distance of concern'[1].

[1] When the bridge receives a detonation event from VMSA, it does some processing before passing the event along to the agents. First of all, it compares the location of the detonation and the location of each agent to decide which agents should be sent the message. In the real world, there is some distance at which you would never even know a detonation of a particular type occurred and a distance at which you wouldn't care enough to physically react. So the bridge doesn't inform agents (continued on next page…)

In JACK, events can:

- be posted by an agent to itself (i.e., internal events),
- represent messages from another agent or percepts from its environment (i.e., external events), or
- represent goals that the agent wants to achieve.

There are two main categories of events in JACK: *normal* events and *BDI* events. Normal events are like events in any other event-driven software paradigm; they represent things that have happened which require an immediate response. BDI events are different in that they produce goals that persist.

When a normal event occurs, at most one plan will be executed to try to handle it. If there is more than one plan that could handle the event, the first plan is always chosen. It is not possible (in JACK) to use additional reasoning to choose one of the applicable plans in another way. If such reasoning is desired, a BDI event should be used instead of a normal event.

When a BDI event occurs and there is more than one plan that can handle it, additional reasoning (meta-level reasoning) may be put in place to specify how one of those plans is chosen. It is also possible for the agent to try another plan after the first one is complete if it has not achieved the desired goal.

There are two categories of normal events in JACK: *Events* (which are posted by the agent to itself) and *MessageEvents* (which are sent to the agent from another agent or its environment). Note that *MessageEvents* can only be sent agent to agent. They can not be sent from an agent to all other agents at once.

BDI events may take the form of *BDIFactEvents*, *BDIMessageEvents*, *BDIGoalEvents*, *InferenceGoalEvents*, or *PlanChoice* events. The first three of these allow 'meta-level' reasoning, which means that the agent can apply additional reasoning to decide which of multiple useable plans it will select and execute. This is in contrast to the normal JACK events which are always handled by the first available plan. When one of these three event types is used and multiple plans (or instances of plans) can handle an event, a *PlanChoice* event is automatically posted, causing execution to switch to handling that event. This results in the selection of one of the useable plans. *BDIFactEvents* are posted by the agent to itself, *BDIMessageEvents* are received from external sources (such as another agent), and *BDIGoalEvents* represent longer term objectives that the agent wants to achieve. For *BDIGoalEvents* events, an additional plan from the plan set will be tried if the first plan fails to achieve the goal, and plan set recalculation can be used so that the available plans are always timely and valid for the current circumstances. And finally, *InferenceGoalEvents* result in all applicable plans being selected and executed, rather than just one.

While JACK does not require explicit identification of goals, it is necessary to identify the goals in order to understand which events should be classified as goal events. A goal can be looked at

---

that won't respond. Secondly, the bridge receives the detonations interaction from VMSA with an exact location. Before sending the message on to any agents that will be concerned with it, the bridge calculates the bearing and range from each of these agents to the detonation, and sends this information with the event. This is done so that the information received by the agents is as close as possible to what they would receive in the real world.

as something the agent wants to achieve and will continue trying to achieve until it believes it has been successful or that there is no way to achieve it.

While the spotter takes responsibility for initial detection of the intruder and sharing knowledge of its approach with the attackers, these activities are not ongoing. Once the intruder is detected, the spotter sends a message to the attackers and its job is done. The spotter does, however, have a goal to protect itself. It will focus on this goal when munitions detonate nearby, and will move to a new destination away from the detonation. The agent will continue with the task of achieving this goal, unless a more important goal arises, such as escaping from a second detonation. It would not make sense for the spotter to get to its second safe location and then modify its course to move away from the first detonation, so that task is dropped.

The attacker has numerous goals it can try to achieve, in addition to its goal to protect itself in the same way the spotter does. It also has goals to 'gather together', 'attack', and 'retreat'. These goals become active based on events that occur or conditions that are met. Each of these goals can be temporarily interrupted by a nearby detonation which puts the 'protect self' goal in focus. After reacting to a detonation, the agent returns to dealing with its previous goal. The 'gather together' goal can also be permanently interrupted by a *VesselCrossedAttackLine_Event* (if that vessel is the intruder) which starts the 'attack' goal. Even if an attacker had not reached the gathering point yet, it would still go in for the attack when the intruder gets to that location.

Despite the various names given to the goals, all of these goals are achieved by repetitive posting of one of three types of events: *MoveChoice_Event, Move_Event* and *Attack_Event*. Thus, only these events are considered *BDIGoalEvents* (which perhaps could have been more accurately named 'events that achieve goals').

The *VesselIDDetermined_Event* is defined as an *InferenceGoalEvent* so that the 'inform attacker' plan which handles it will be executed for every attacker.

The remaining events are either *Events* (if posted internally), *MessageEvents* (if posted externally from another agent or the bridge), or *PlanChoice* events (if used to choose between more than one plan). Table 11 shows which event types have been used to implement the previously described events in JACK.

Table 11: Event types for events in swarming scenario

Event Type	Events
Event	*StartSimulation_Event* *VesselCrossedAttackLine_Event* *EndAttack_Event*
MessageEvent	*MunitionDetonation_Event* *IntruderDetected_Event*
InferenceGoalEvent	*VesselIDDetermined_Event*
BDIGoalEvent	*MoveChoice_Event* *Move_Event* *Attack_Event*
PlanChoice	*ChoosePoint_Event*

3.2 Plans

A plan is executed in response to an event. Plans are made up of a sequence of actions. Each plan can only handle one type of event, but more than one plan can handle that same type of event. Each plan has a *relevant* filter to determine which plans are relevant and a *context* filter to determine which plans are applicable.

A plan is said to be relevant if it specifies that it handles the event and if the parameters of the event match the pattern specified in the *relevant* method. Consider the *Move_Event* which has parameters EntityName, Position, Course, and Speed. When the *Move_Event* is posted, it is not necessary for all of the parameters to have values. This event can result in using one of four plans which will make the entity move in one of four ways: towards an entity, towards a point, along a bearing, or continuing with the current course and only changing speed. Which plan will be selected depends on which parameters have values in them. For example, the *relevant* filter on the *MoveToEntity_Plan* which handles the *Move_Event* is 'EntityName <> null'. Similar filters are used for the other plans (and are described in Table 12).

The *context* filter is applied to all relevant plans and is not quite as straight forward. There are two cases where the context method is used: (1) to determine whether or not the plan is applicable based on the agent's current beliefs and (2) to generate multiple instances of a plan based on the agent's current beliefs. These will be illustrated with examples.

An example of checking plan applicability based on current beliefs can be seen in how the *Attack_Event*(EntityName) is handled. In order to attack an entity, the attacker must be within weapon range of the entity. So there are two plans which handle the *Attack_Event*: *GetInRange_Plan*(EntityName) and *Fire_Plan*(EntityName). For the *GetInRange_Plan*, the *context* method would be something like: *RangeTo(*EntityName*)* > *GunRange*, and the *context* filter for *Fire_Plan* could be: *RangeTo(*EntityName*)* <= *GunRange*. *GunRange* is a value obtained from the entity's *Weaponry* beliefset, and the range from the attacker to *EntityName* can be found in the entity's *Detections* beliefset. Since data from beliefsets is required, this filter is part of the context rather than the relevant clause.

An example of generating multiple instances of a plan based on the agent's current beliefs is seen in how the *MoveChoice_Event* that is posted from within the *Gather_Plan* is handled. There is one relevant plan for this event: *MoveChoicePoint_Plan*. The context of this plan type causes a plan instance to be created for each entry in the *Gathering Areas* data set. In this case, there are two positions, so two plan instances are created. When there is more than one applicable plan instance for a given event, JACK posts an event of type *PlanChoice*. For this situation, the *ChoosePoint_Event* is posted. This event is handled by the *ChooseClosestPoint_Plan* which compares the distance from each of these points to the agent's location and selects the closest point. This point is returned to the *MoveChoicePoint_Plan* which posts a *Move_Event* to move the entity towards the closest point.

The plans that were used to implement the scenario previously discussed are outlined in Table 12. When appropriate, the table indicates when the plans are relevant and applicable (through the relevant and context filters), the purpose of the plan (the body), the events the plan posts, and the beliefsets the plan uses.

Table 12: Plans to Handle Events of Scenario

Event Handled	By Plans
StartSimulation_Event	*ActInnocent_Plan* body: Move around slowly. (This is the default / fall-back behaviour for the attackers.)
VesselIDDetermined_ Event (EntityName)	*InformAttackers_Plan* context: If EntityName is in the *Intruders* beliefset, generate an instance of this plan for every name in the *RedTeamNames* beliefset. body: Send 'intruder detected' message to attacker. posts: *IntruderDetected_Event*(EntityName) uses: *Intruders, RedTeamNames*
IntruderDetected_Event (EntityName)	*Gather_Plan* body: Move towards one of the gathering locations until the distance between the agent and any gathering location defined in G*atheringAreas* is less than the Range defined in the beliefset for that location. posts: M*oveChoice_Event(*Point*)* uses: *GatheringAreas, Self*
MoveChoice_Event (ChoiceType)	*MoveChoicePoint_Plan* relevant: ChoiceType = Point context: Generate an instance of this plan for each point in the *GatheringAreas* beliefset. A *ChoseClosestPoint_Event* is automatically posted to determine which point the agent should head for (i.e., which plan instance should be executed). body: Post move event to move the entity towards the closest gathering point. posts: *Move_Event* uses: *GatheringAreas, Self*
ChoosePoint _Event	*ChooseClosestPoint_Plan* body: Compare distance from self to all gathering points and return the gathering point that is closest. uses: *GatheringAreas, Self*

Move_Event (EntityName, Position, Course, Speed)	*MoveToEntity_Plan* relevant: EntityName <> null body: If Speed <> null, move towards EntityName at a speed of Speed. If Speed = null, move towards EntityName while maintaining current speed. uses: *Detections* *MoveToPoint_Plan* relevant: Position <> null body: If Speed <> null, move towards Position at a speed of Speed. If Speed = null, move towards Position while maintaining current speed. uses: *GatheringAreas* *MoveInDirection_Plan* relevant: Course <> null body: If Speed <> null, move along a course of Course at a speed of Speed. If Speed = null, move along a course of Course while maintaining current speed. *ChangeSpeed_Plan* relevant: EntityName = null, Position = null, Course = null, Speed <> null body: Change speed to Speed.
VesselCrossedAttack Line_Event (EntityName)	*Attack_Plan* context: EntityName is in the *Intruder* beliefset. body: Attack EntityName until it has a damage level >= Damage (as defined in *MissionSuccess* beliefset) OR Ammunition = 0 OR number of team members destroyed > TeamLossTolerance. posts: *Attack_Event* uses: *Detections*, *Weaponry*, *Intruders*, *RedTeamNames, TeamLossTolerance, MissionSuccess*

Attack_Event (EntityName)	*GetIntoRange_Plan* context: *RangeTo*(self, EntityName) >= GunRange body: Move towards EntityName until the agent is within GunRange of EntityName. posts: *Move_Event* uses: *Detections*, *Weaponry*
EndAttack_Event	*Retreat_Plan* body: Stop firing. Move away from all frigates that are currently seen (i.e., in the *Detections* beliefset). Remain at current position as long as no frigates can be seen[3]. posts: *Move_Event* uses: *Detections*
MunitionDetonation_Event (EntityName, MunitionType, Range, Bearing)	*MoveToSafety_Plan* body: Move away from the detonation until the agent is SafeDistance from the detonation. posts: *Move_Event* uses: *DetonationResponses*

3.3 Beliefsets

In JACK, the data that is used by the agents to make decisions is held in their beliefsets. Beliefsets are represented using 'tuple relations', which essentially the non-developer can think of as tables. Each beliefset has a name, zero or more key fields, and additional value fields. The key field, or combination of key fields, is what makes each entry unique. The value fields hold the information the agent needs to know about the key.

JACK ensures that the information in the beliefset is logically consistent. So, if a new belief contradicts an old belief, the old belief is removed before the new belief is added.

Beliefs can be either true or false. For example, if the 'negotiator' spotters were modelled, they could have a beliefset that keeps track of all the intruders. When they detect a tanker that is not an intruder, they could add a *false* entry to their beliefset indicating that it is not true that the tanker is an intruder. Of course, these beliefs are not absolute truths – they are what the agent *believes* to be true. It is possible that the tanker could indeed be an intruder.

[2] For VBE-E, this affiliation change was a manual process performed for each entity through the JSAF interface. This affiliation change was used to determine which visual model of the entity (without guns, or with guns) was shown to the UAV operator when in detection range.

[3] Note that this plan is never complete. It remains active until the simulation ends, since even after an entity has moved away from the frigates, it is possible that the frigates will move towards the agent such that it once again detects them.

JACK uses either Open World or Closed World semantics to manage its beliefsets [4]. In Closed World relations, a beliefset includes a complete list of all possible information relevant to that beliefset. For example, a beliefset for a tic tac toe game could include a belief for all nine boxes that indicates whether or not they are still empty. Closed World semantics normally do not make sense for real-world applications.

On the other hand, with Open World relations, only the beliefs that are known to the agent are included in the beliefset. If something could be represented in a beliefset, but is not in the beliefset, this means the agent does not know whether that statement is true or false.

The Open World model has been used for modelling the red forces discussed in this paper. However, it is not clear that there is any reason to have any false beliefs to handle this problem. In future extensions to this simplified problem, the benefits of using false statements may become clearer. In any case, it is perfectly acceptable to only store true statements in this Open World model. The fact that the majority of the data that the agents place in their beliefsets comes from a non-agent simulation may be influencing this choice.

The final issue that must be mentioned before defining the beliefsets for this project is the accessibility of information in a beliefset. In JACK, beliefsets can provide 'private', 'agent' or 'global' level access. Private-level beliefsets can only be *read* by the agent itself (e.g., an attacker), agent-level beliefsets can be read by all agents of a particular type (e.g., all attackers), and global-level beliefsets can be read by all agents of all types (e.g., all attackers and spotters). *Writing* and *modifying* is more restricted. Agents can only modify beliefs in their own, private-level beliefsets. All data within agent-level or global-level beliefsets must be set at the beginning of a simulation and remain unchanged throughout the simulation. These beliefs are meant to represent things that all attackers would know (e.g., the definition of a successful attack) or that all attackers and spotters would know (e.g., the ID of the intruder). Data that is specific to the individual and changes throughout the scenario must be represented as private data (e.g., the sensor tracks that an attacker or spotter holds). It should also be noted that this project is being implementing using the JACK Agents only. Future efforts may investigate the use of JACK Teams where the concept of 'team data' is introduced.

In Section 2.3, the beliefsets were partially specified. It seemed natural to identify what information the agents needed while the problem was being decomposed. Now it is necessary to make them a little more JACK-like.

The following tables define the beliefsets and any data initialization that is required. They also indicate whether or not the beliefset will be modified during execution. For the agent-level and global-level beliefsets, there is no 'Modify' column, since these beliefs do not have the option of being modified.

3.3.1 'Private' Beliefsets

Attackers must know about their weapons and ammunition as shown in Table 13. This information comes from and is updated by VMSA. The spotter does not have weapons so it does not require a weapons beliefset.

Table 13: Private-Level Beliefset for Attackers Only

Beliefset name	Key field(s)	Value field(s)	Initialization	Modify
Weaponry	WeaponName	WeaponType WeaponRange WeaponStatus AmmunitionType AmmunitionCount	None (this information is passed from VMSA).	True

Private beliefsets that are needed by both spotters and attackers are shown in Table 14.

Table 14: Private-Level Beliefsets for Both Spotter and Attackers

Beliefset name	Key field(s)	Value field(s)	Initialization	Modify
Detections	EntityName	Domain Type ID Bearing Range Damage Time	None	True
Muntion Detonation Responses	Ammunition Type	SafeDistance	SafeDistance for all possible ammunition types that may detonate/land near an attackers/spotter.	False
Self	OwnName	Latitude Longitude Course Speed	None (this information is passed from VMSA).	True
TeamLoss Tolerance	Limit		The number of team members that must die before the entity retreats.	False

All agents have visual capabilities and keep track of their detections in their *Detections* beliefsets. These detections are generated by VMSA and passed through the bridge.

The *MunitionDetonationResponses* beliefset tells each agent how to respond to a particular type of detonation. Each agent may respond differently. This information is fixed at start-up.

The *Self* beliefset represents all the information the agent needs to know about itself: where it is, where it is going, and how fast. This information is provided by VMSA.

The *TeamLossTolerance* beliefset indicates how many team members the agent can tolerate losing before retreating. This number may vary from agent to agent and is fixed at start-up.

3.3.2 'Agent' Beliefsets

Agent-level beliefsets for attackers are shown in Table 15. The attackers need to know how 'mission success' is identified so that they will know when they should retreat. They also need to know the gathering locations when they are preparing for an attack, and the attack line that the intruder must cross to trigger the attack. This information is fixed at start-up.

Table 15: Agent-Level Beliefsets for Attackers Only

Beliefset name	**Key field(s)**	**Value field(s)**	**Initialization**
MissionSuccess	EntityName	Damage	Intruder name and desired damage.
GatheringAreas	LocationName	Latitude Longitude Range	Two gathering points and ranges.
AttackLine	PointName	Latitude Longitude TestPoint	Two points to mark the line, one on either side of the strait and a third 'test point' to mark one side of the line.

3.3.3 'Global' Beliefsets

Global-level beliefsets for the spotter and attackers are shown in Table 16. Both the spotter and attackers need to know who the intruders are in order to detect and identify them. They both need to have a list of all the attackers as well; the spotter needs to know who to send the 'intruder found' message to and the attackers need to monitor how many team members have been destroyed.

Table 16: Global-Level Beliefsets for Both Spotters and Attackers

Beliefset name	**Key field(s)**	**Value field(s)**	**Initialization**
Intruders	EntityName	Type ID	A list of intruders, including frigates.
RedTeamNames	EnityName		Names of all attackers.

3.3.4 Time and Priority Task Management Beliefset

A final beliefset is required to handle time management and the prioritization of the agent's tasks. This beliefset is used to (1) keep the agent simulation synchronized with the time-managed VMSA simulation, and (2) tell the agents which of its goals are most important and when a goal is no longer relevant. While some agent software development tools may include a mechanism for handling goal suspension and dropping, this is not a built-in feature of JACK. Thus, each agent is given a private-level *Tasks* beliefset (Table 17) with Active and Drop fields which are used by the agent to determine which task should be executed first and which task(s), if any, should be deleted (i.e., those marked as dropped). It also includes two priority fields (Priority and SubPriority) and a Status field which are used by the bridge to determine how to set the Active and Drop fields.

Table 17: Tasks Beliefset for Spotters and Attackers (Private-Level)

Beliefset name	Key field(s)	Value field(s)	Initialization	Modify
Tasks	TaskID	Priority SubPriority Status Drop Active	None	True

The Status field must be set by the plan that initiates the task to achieve a goal. There are five plans which trigger a new task: *ActInnocent_Plan, Gather_Plan, Attack_Plan, Retreat_Plan,* and *MoveToSafety_Plan.* At the end of each of these plans, there must also be a statement to delete the task from the *Tasks* beliefset entirely.

For the scenario at hand, the "act innocent" task provides the default agent behaviour and has the lowest priority. However, it is never dropped by the addition of another task. The gather task can be dropped by the attack task (or the retreat task, although this is unlikely to happen), but not the move to safety task. The attack task is removed from the task list by itself when one of the conditions for calling off the attack occurs (i.e., intruder is damaged, team loss is high, or ammunition is gone). Thus, no other task causes the attack task to be dropped[4]. The retreat task is not dropped until it can no longer see a frigate, so no other task causes the retreat task to be dropped either. The move to safety task is the highest priority task (and can therefore suspend any other task) and can be dropped by another task of the same priority.

A list of priorities and status values that can be used to satisfy the statements above is shown in Table 18. Note that these work for both the spotter and attackers.

[4] It is not necessary for the 'retreat' task to cause the 'attack' task to be dropped, since the 'retreat' task is only initiated by the completion of the 'attack' task (which happens as a result of intruder damage, team loss, or depleted ammunition). The plan that initiated the 'attack' task will delete this task from the Tasks beliefset before the 'retreat' task is ever added to it.

Table 18: Task Priorities and Status

Task	Priority	Status
act innocent	0	0
gather	1	1
attack	1	0
retreat	1	0
move to safety	2	1
`Status: 0-Never dropped` `1-Dropped by task of same priority`		

The issue of priority task management is discussed in more detail in [10].

3.4 Agents

In the JACK Agent Manual, it states that JACK agent declarations need to include the following [4]:

- beliefsets the agent can use,
- events (both internal and external) that an agent is prepared to handle, and
- plans the agent can execute.

From the previous sections of this paper, it is now possible to identify these elements for each agent.

While there are really only two different agent types required by this project, a third agent type 'RedForce' will be introduced here which encapsulates all of the things in common between the spotter and attacker agents (Figure 4). Both the spotter and attacker agents will then be descendants of the RedForce agent. This makes the code development more straight forward.

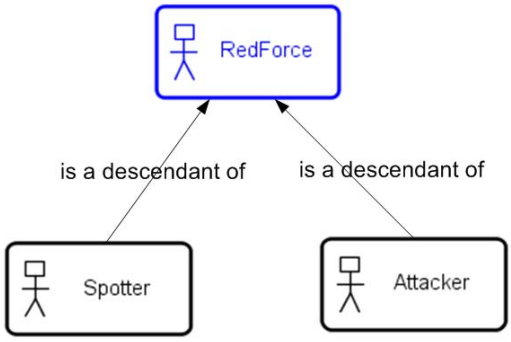

Figure 4: The Red Force agent

Table 19 lists the beliefsets, events, and plans used by the spotter and attacker agents (i.e., the RedForce agent).

Table 19: RedForce Agent Beliefs, Events, and Plans

Beliefsets	**Events Handled**	**Events Posted**	**Plans**
Detections *Intruders* *MuntionDetonationResponses* *RedTeamNames* *Tasks*	*Move*	*Move*	*ActInnocent* *MoveToPoint*

Table 20 lists the events and plans used by the spotter agent in addition to those described for the RedForce agent.

Table 20: Spotter-Specific Beliefs, Events, and Plans

Beliefsets	**Events Handled**	**Events Posted**	**Plans**
(none)	*VesselIDDetermined*	*IntruderDetected* *VesselIDDetermined*	*InformAttackers*

Table 21 lists the beliefsets, events and plans used by the attacker agent in addition to those described for the RedForce agent.

Table 21: Attacker-Specific Beliefs, Events, and Plans

Beliefsets	**Events Handled**	**Events Posted**	**Plans**
AttackLine *GatheringAreas* *Self* *TeamDestroyed* *Weaponry* *MissionSuccess* *TeamLossTolerance*	*Attack* *ChoosePoint* *EndAttack* *IntruderDetected* *MoveChoice* *VesselCrossedLine*	*Attack* *ChoosePoint* *EndAttack* *MoveChoice* *VesselCrossedLine*	*Attack* *ChooseClosestPoint* *Fire* *Gather* *GetIntoRange* *MoveChoicePoint* *MoveInDirection* *MoveToEntity* *Retreat*

4 Design Diagrams

The following diagrams were developed using the JACK design tool. However, some of the (repetitive) links have been removed from the diagrams to make them more readable. The diagrams in this section are meant to give the user an overview of how the various beliefsets, plans and events are related. This information was described verbally in the previous sections, but these diagrams provide a clearer idea of the relationships. They are also used by JACK to automatically generate some of the code that is required at implementation time. The logic has been broken into six pieces for greater readability. Each diagram deals with a particular aspect of the scenario, showing the events handled and posted and the plans and beliefsets used.

Unfortunately, the JACK design tool has been built such that some of the labels on the directed lines are counter-intuitive. The link from an event to an agent or an event to a plan is labelled with the term 'handles', when in fact the agent and the plan handle the event (not the other way around, as the diagrams seem to suggest). This should be kept in mind when looking at the design diagrams in this section.

The diagrams presented are as follows:

- the red force acting innocent,
- the spotter watching for intruders,
- the attackers gathering,
- the attackers attacking,
- the attackers retreating, and
- the red force moving to safety.

4.1.1 The Red Force Acting Innocent

Figure 5 shows how the spotter and attackers respond to the *StartSimulation_Event*. This event is handled by the *ActInnocent_Plan* which tells the agents to move around while they wait for something interesting to happen. This initial plan is very simple and tells an agent to move along its initial course (which can be obtained from the *Self* beliefset), randomly[5] changing direction, and never moving further than a pre-specified distance from its position at the start of the behaviour[6]. Each time a course change is required, the *ActInnocent_Plan* posts a *Move_Event* which is handled by the *MoveInDirection_Plan* which sends a request to the VMSA motion federate via the bridge to change the agent's course. This is the behaviour that the agent will always fall back into in the absence of anything more important to do.

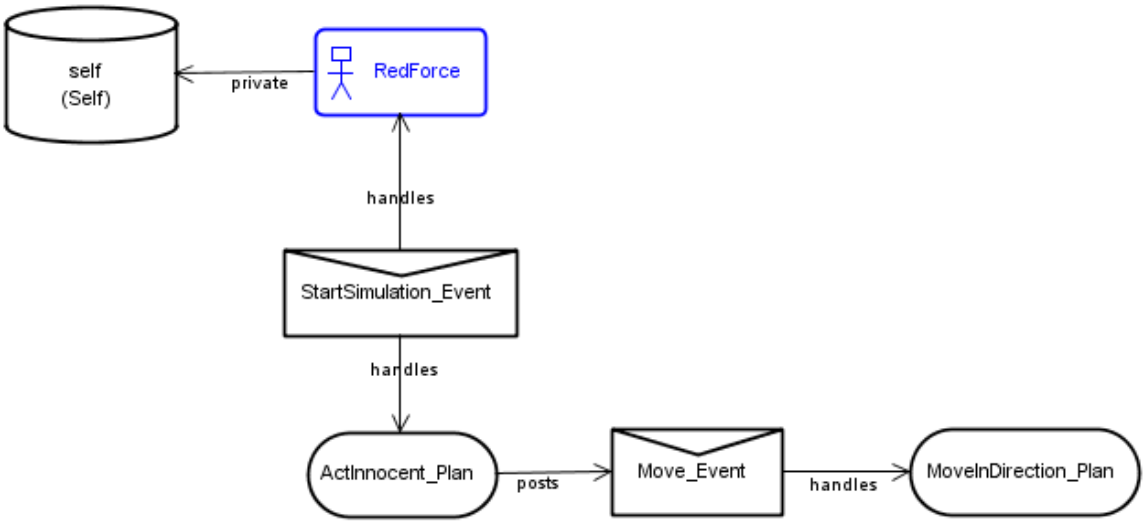

Figure 5: The Red Force Acting Innocent

[5] A random number seed is used so that these movements are repeatable.
[6] This is done to keep the agents from driving the entities onto land.

4.1.2 The Spotter Watching for Intruders

Figure 86 shows the spotter watching for intruders and informing the attackers if it detects one. The *Detections* beliefset contains a list of all the vessels that the spotter can currently see. When the ID of a vessel is first determined, the *VesselIDDetermined_Event* is posted. This event is handled by the *InformAttackers_Plan* only if the ID matches the ID of a non-frigate intruder in the *Intruders* beliefset. If there is a match, an instance of the *InformAttackers_Plan* is generated for each name in the *RedTeamNames* beliefset. Since the *VesselIDDetermined_Event* is an *InferenceGoalEvent*, each instance of the plan is executed, which in turn posts the *IntruderDetectedBySpotter_Event* to each of the attackers.

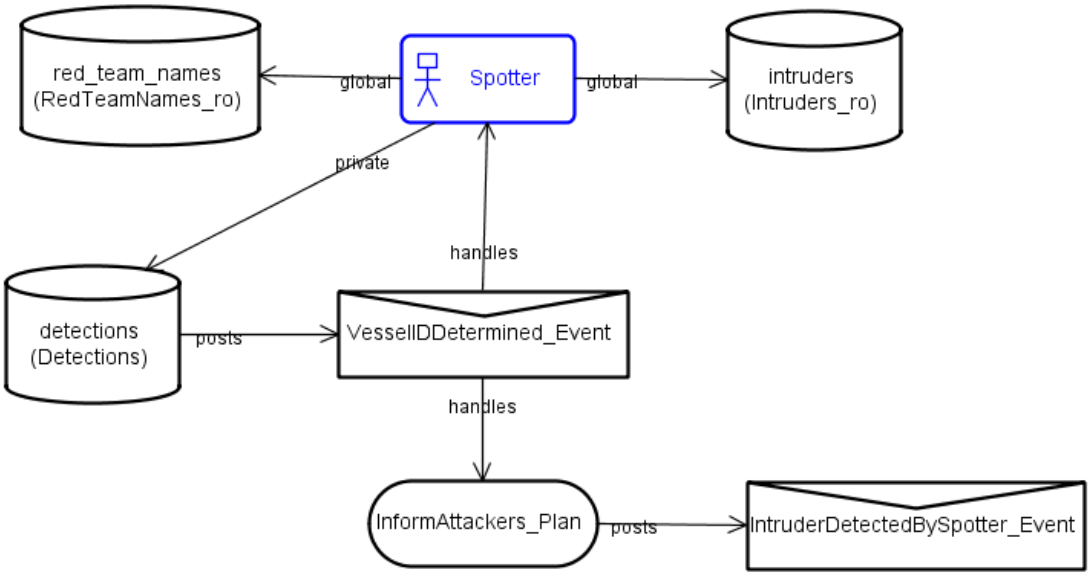

Figure 6: The Spotter Watching for Intruders

4.1.3 The Attacker Gathering

Figure 7 shows the gathering phase. The attacker receives the *IntruderDetected_Event* from the spotter. The *Gather_Plan* handles this event by posting a *MoveChoice_Event* over and over until the attacker reaches one of the gathering areas defined in *GatheringAreas*. This event is handled by the *MoveChoicePoint_Plan* which creates an instance of itself for each possible gathering point in *GatheringAreas*. Since there are two such points, this means that there are two plans that the agent must choose between (one plan will guide it to one point and the other plan will guide it to the other point). To handle this decision, a *ChoosePoint_Event* (a *PlanChoice* event) is posted which is handled by the *ChooseClosestPoint_Plan* which compares the distance from the attacker to the two gathering locations and chooses the closest one. This choice is returned to the *MoveChoicePoint_Plan* which then posts a *Move_Event* that gets handled by the *MoveToPoint_Plan* which (finally) sends a request to the VMSA motion federate via the bridge to move the agent towards the closest point.

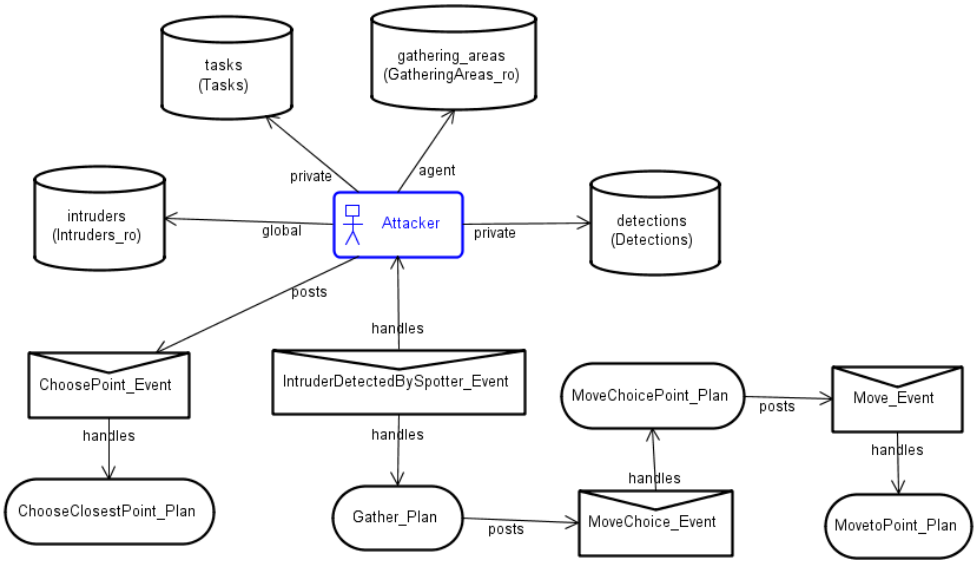

Figure 7: The Attacker Gathering

If the attacker is interrupted while heading to a gathering point (such as when a munition detonation occurs and causes it to move off its current track), it will re-evaluate which gathering point is closest when it returns to its gathering mode, and it will head towards the closest point at that time. Thus, it is possible that the point that the attacker initially heads for will not be the point that it arrives at.

4.1.4 The Attacker Attacking

Figure 8 shows the attacker agent's attack phase. When a vessel crosses the attack line, the *VesselCrossedAttackLine_Event* is posted. This event is handled by the *Attack_Plan* only if the vessel is the intruder. The *Attack_Plan* posts the *Attack_Event* over and over until one of the conditions for calling off the attack has been met. The *Attack_Event* is handled by one of two plans: *GetIntoRange_Plan* and *Fire_Plan*, depending on whether or not the attacker is within weapon range. The *GetIntoRange_Plan* further posts a *Move_Event* which is handled by the *MoveToEntity_Plan* which moves the attacker closer to the intruder through requests to the VMSA motion federate via the bridge.

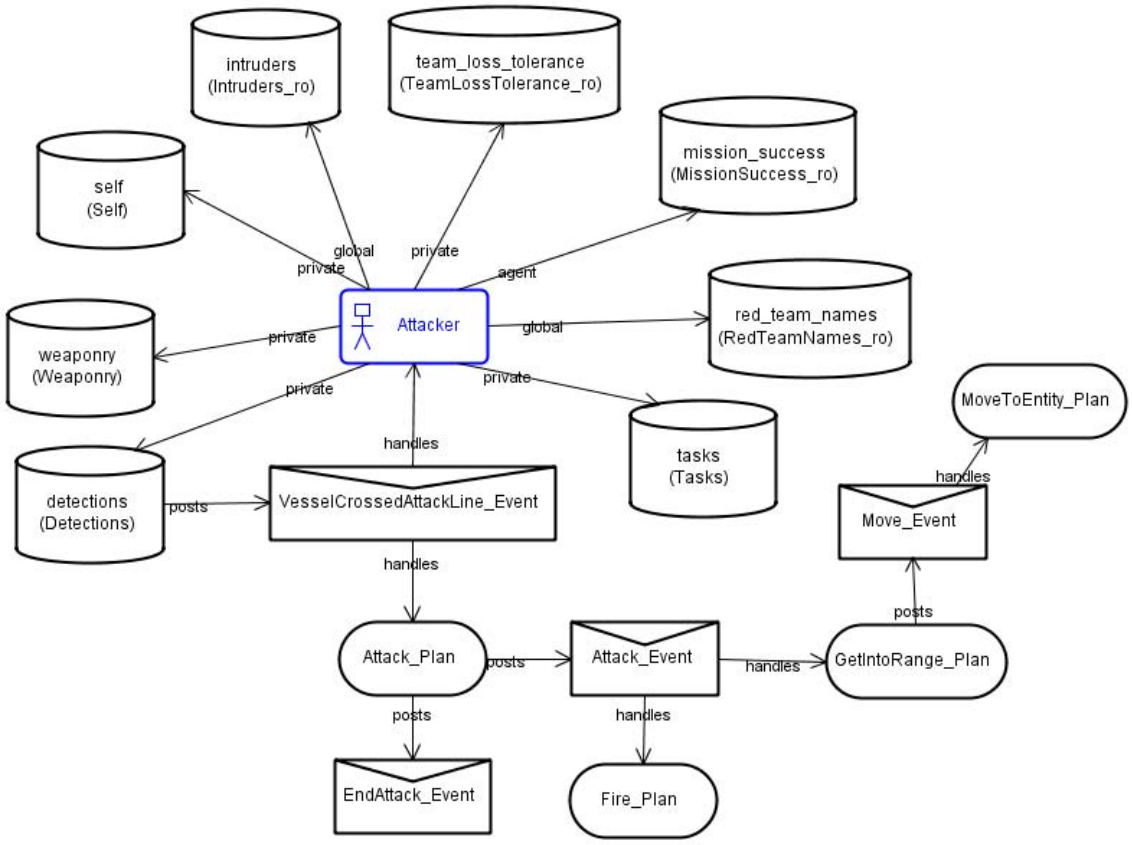

Figure 8: The Attacker Attacking

The *Fire_Plan* sends weapon fire requests to the VMSA gun federate via the bridge. When one of the conditions for terminating the attack has been met (i.e., the intruder is damaged 'enough' according to the Intruders, Detections[7] and MissionSucces beliefsets, the team loss has exceeded the attacker's tolerance according to the TeamLossTolerance, RedTeamNames and

[7] Recall that damage levels are found in the Detections beliefset.

Detections beliefset, or the attacker is out of grenades (according to the Weaponry beliefset)) the *Attack_Plan* posts a final event before its completion: *EndAttack_Event*.

4.1.5 The Attacker Retreating

Figure 9 shows the attacker agent's retreating phase. The *EndAttack_Event* is handled by the *Retreat_Plan* which queries the *Detections* beliefset to determine which (if any) frigates the attacker can see, and calculates the direction in which the attacker should retreat using a proximity-weighted average of the directions opposite the frigates (see examples in Figure 10). The *Retreat_Plan* posts a *Move_Event* over and over until the attacker can no longer see either of the frigates. This event is handled by the *MoveInDirection_Plan* which sends a request to the VMSA motion federate via the bridge to move the agent in the chosen direction.

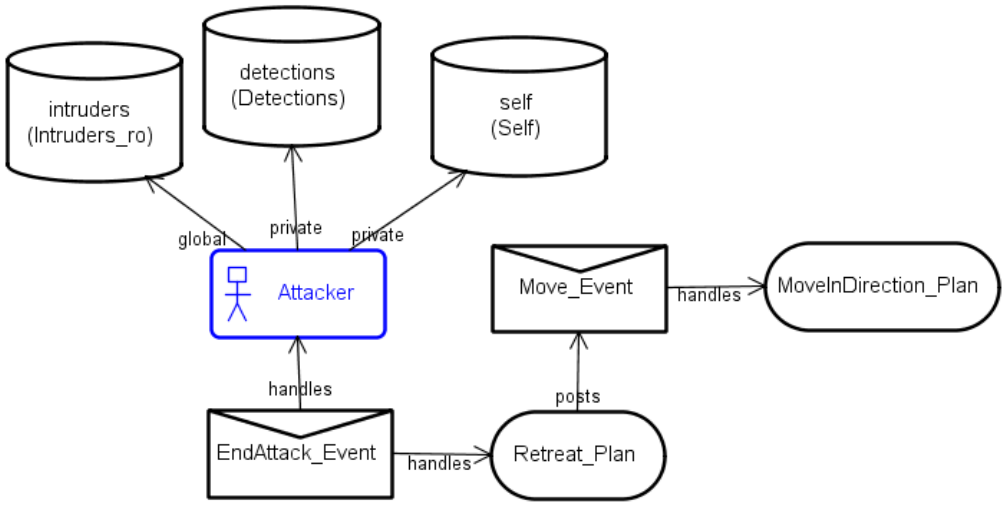

Figure 9: The Attacker Retreating

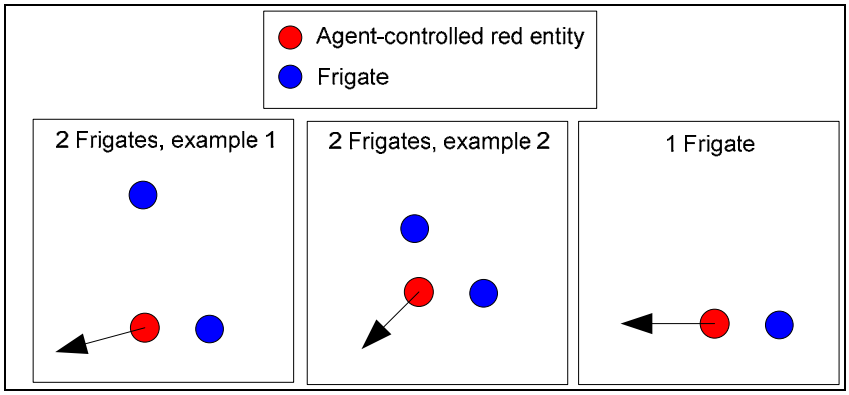

Figure 10: The Attacker's Retreat Direction

At each simulation time step, the *Retreat_Plan* re-evaluates the direction that the attacker should make its escape in. So, as the frigates themselves move and the bearings from the attacker to the frigates change, the attacker's course is modified accordingly.

4.1.6 The Red Force Moving to Safety

Figure 11 shows how the spotter and attackers respond to *MunitionDetonation_Event*s. The event is handled by the *MoveToSafety_Plan* which posts the *Move_Event* over and over to make the agent alter course (which requires knowing the agent's current course which is obtained from the *Self* beliefset) until the agent has moved a particular distance (as specified in the *MunitionDetonationResponses* beliefset) away from the munition detonation location. The *Move_Event* is handled by *MoveInDirection_Plan* which sends a request to the VMSA motion federate via the bridge to temporarily move the agent along this modified course.

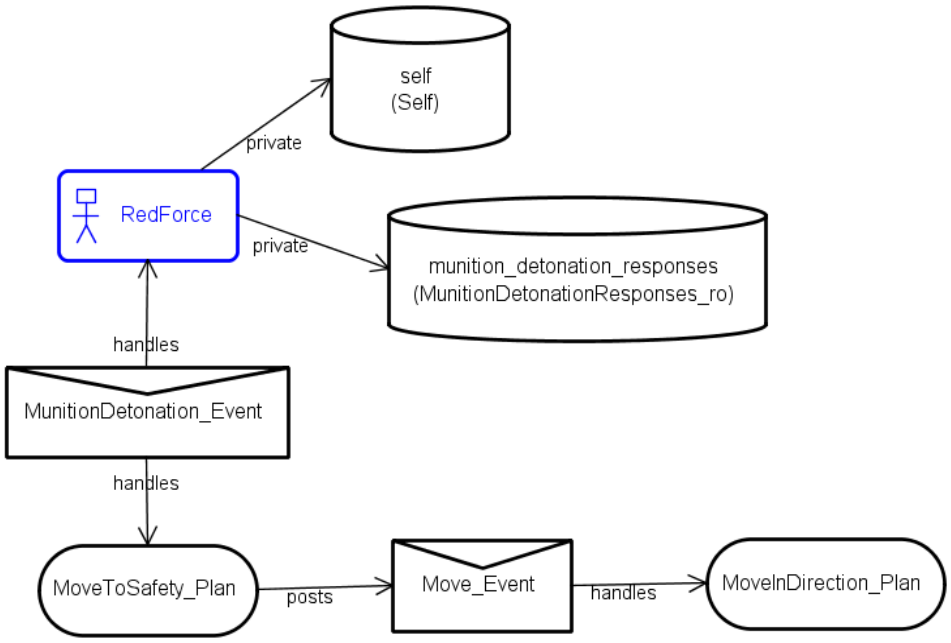

Figure 11: The Red Force Moving to Safety

5 The Results

The design discussed in Sections 3 and 4 was implemented in JACK. Since all of the JACK agents, beliefsets, events and plans and the relationships amongst them were identified during the design phase, the JACK implementation was fairly straight forward. Aiding in this was the fact that the JACK design tool was used to create the diagrams in Section 4, providing a head start on the implementation process. For example, by right-clicking on any of the events or plans in the design diagrams, a screen for editing that event or plan appears. Any information that could be obtained from the diagrams, such as what events a plan posts, was already (automatically) written in the code. And, despite the numerous events and plans, the details that had to be coded within each of them were fairly simple. The bigger challenge was developing the software to bridge JACK and VMSA, such that data from the VMSA simulation could meaningfully find it is way into the appropriate JACK agent beliefsets, and such that the agent's requests for movement and gun fire were heard and correctly interpreted by the VMSA simulation. A technical description of this software bridge can be found in [6].

It is worth noting as well that designing the JACK agents, implementing them in JACK, and building the JACK-VMSA bridge has taken about four person-months (not including the time to set up the contract for technical support and bridge development), where neither person was experienced with the JACK software. Some of this time was spent handling unexpected technical issues that are now resolved and should not pose an issue in the future. In any case, this is a relatively short period of time and hopefully gives an idea of how much could be accomplished with the appropriate amount of time and effort. This should also be remembered when comparing the results of this implementation to other long standing software systems such as JSAF.

Through the creation of this scenario in JACK and the development of the software bridge, the potential for agents to control red force and/or neutral entities in VMSA simulations for experimentation has been demonstrated. Their utility falls primarily into three categories: entity realism, scenario development, and scenario repeatability.

5.1 Scenario Realism

Agents, and therefore the entities they control, can exhibit both reactive and proactive behaviours. The reactive behaviour allows entities to respond to occurrences in their environment (e.g., a nearby detonation causes a temporary change in the entity's course), to messages from other entities (e.g., when the spotter informs the attackers of the detection of an intruder, they immediately change their behaviour), or to a change to data in their beliefsets (e.g., when the spotter's *Detections* beliefset shows an ID which matches an ID in the *Intruders* beliefset, it immediately informs the attackers). The proactive behaviour allows an entity to have a longer term goals (e.g., to get within firing range of the intruder), and even after being interrupted by higher priority events such as nearby gun fire, it is able to return to that goal and try to satisfy it. This goal-oriented behaviour of the attacker is seen various times in the scenario presented: moving to the closest gathering point, fleeing from weapon fire, attacking the intruder, and retreating so that it cannot be seen by either of the frigates. This proactive behaviour makes the entities appear focused and aware of their surroundings. Their ability to make context-sensitive decisions (through the use of context filters in the plans) enhances this. For example, when the

attackers have moved into their attack mode, they know how far they are from the intruder as well as the limitations of their weapons, and they will not fire on the intruder if they are not within range.

This agent solution has another benefit in terms of realism over the set-up used for VBE-E with human JSAF operators advancing the behaviours of the red force entities. For VBE-E, the JSAF operator was given a map display that showed the location of *all* entities – red, green, and even blue. While the intention of the JSAF operator was to *not* take advantage of this 'god's eye view' when guiding the red force, having complete knowledge of where the target entities were located was somewhat difficult to overlook. While JSAF could have been set up so that the blue and green entities could not be seen, and such that only sensor detections were available for making the decisions to control the red force, monitoring the detections of 8-10 red force entities was just not possible for one person. On the other hand, the agent-controlled entities are not all-knowing. They base their decisions on the information in their beliefsets, in particular, their *Detections* beliefset. The information in their *Detections* beliefset comes from the sensor models in VMSA. For the scenario discussed in this paper, their *Detections* beliefset simply contained visual sensor data (i.e., a list of entities that could by physically seen by a human, based on the proximity and type of vessel), so any vessels beyond sight remain unknown to the attackers.

A final point can be made in terms of improved realism. For VBE-E, the time-managed VMSA simulation was often unable to maintain real-time speed while networked to other participating countries. JSAF, unlike VMSA, is not time-managed. JSAF runs at real time, regardless of how fast the VMSA portion of the simulation is running. For VBE-E, JSAF was controlling the red and neutral entities and VMSA was controlling (through human operator input) the blue force. It frequently appeared to the operators that the red and neutral forces were moving much faster than they should be (and relatively speaking, they were). JACK agents, on the other hand, are being guided by the VMSA simulation clock and will advance as slowly or quickly as the rest of the VMSA simulation and will not result in this type of confusion.

5.2 Scenario Development

While the concept of using agents for scenario control has been demonstrated, it would be unreasonable to state this early on that agents will make scenario development easier or faster. If the particular application requires a swarm attack scenario, clearly the process will be much simpler than it would be for an entirely different scenario, given that many of the necessary constructs would already exist. It will take time to build up a wide variety of beliefsets, plans and events that the user can choose from when designing a scenario. It is hoped that the other benefits of using agents will be desirable enough to drive the continued development of this library of agent components, such that the benefits of agent-controlled behaviour can eventually be realized for scenario development as well. In the longer term, the aim is to allow scenario developers to define scenarios at a higher level, so that, for example, they specify the location of fishing ports and fishing grounds, rather than the locations of each fishing boat. The details of where the entities are, where they go and what they do would be dealt with by the agents.

5.3 Scenario Repeatability

It is sometimes easy to overlook the importance of scenario repeatability, *until* it comes time to do the analysis. For VBE-E, the JSAF operators used some general rules for when to make the FIACs head to their collecting area, when to make them head for the target vessel, whether they should attack from the front or the back, and when to change each attacker's classification to hostile (from neutral). However, it was hard to stick to these rules at run-time, and ad hoc decisions were inevitably made to deal with unexpected issues, such as simulation slow-down. In contrast, agent-controlled entities will exhibit the same behaviours each time the scenario is run. Given the exact same circumstances a second time, the agent will perform identically since it will have the exact same information available to guide its decisions. It is true, however, that the agent's behaviour will be influenced by the choices of the human operators (which is necessary in order for their behaviour to appear realistic). For example, during the attack phase the attackers move towards the intruder so that they can get within grenade-launching range. If the (human-controlled) intruder heads south, the attackers will head south; alternatively, if the intruder had headed north, the attackers would have headed north. Any lack of repeatability in the experiment is inherent to the human-in-the-loop element of the simulation, not the agents.

In addition to the benefits listed above, JACK agents and VMSA make a nice overall fit. The agents are being used to do exactly what agents are meant to do – make decisions, just as human operators would. They are not being used to model sensors, motion, weapons, or any other physical components; this is handled entirely by VMSA. This makes the agent design quite 'clean' and limits the amount of code that must be written to implement the agents. Perhaps more importantly, it means that an agent-controlled entity and a human-controlled entity of the same type (e.g., a frigate) would use the same physical models. This eliminates any issues of an uneven playing field that could result from the use of two different models of the same sensor or same weapon. It also means that all agent-controlled entities immediately have access to models for all of the sensors and weapons already developed in VMSA.

6 Future Work

The scenario that has been outlined in this document has been kept intentionally simple, while still allowing for the demonstration of JACK's key features. It demonstrates the use of JACK message, inference and goal events, relevance and context filters in plans to guide action selection, and meta-level reasoning to choose between multiple applicable plans. It would be nice, however, to expand on this scenario and explore some more complicated behaviours. Some questions that could be explored include 'What if…':

- individual personality traits (e.g., anger/fear) were tracked and used to influence decision making?
- the attackers had multiple weapons with different effectiveness ranges?
- the attackers had a preference for firing at the intruder while in close proximity to neutral traffic?
- the attackers could also fire at the frigates?
- the intruder turns around and exits the strait? What should the attackers do?
- the red force could share sensor tracks with each other? How could automatic track sharing and fusion be handled?
- the spotter is killed? Could an attacker take over the spotter's role?
- there were *two* (non-frigate) intruders?

There are many questions such as these that could be considered in order to make the existing scenario more interesting and to further explore the power of JACK agents.

Behaviours for all neutral traffic should be developed. For the initial demonstration of this project, the small neutral fishing boats were given initial positions and speeds and the 'act innocent' behaviour was used to control their movements (so in fact these were controlled by agents as well). Of course, in the real world, all small boats will not act the same. The larger boats (the tanker, merchant, and tug boats) in this scenario were given initial positions with courses and speeds that they would maintain throughout the scenario (so there was no agent control), unless otherwise directed by a human operator through the C2 system. In the future, large vessels which are expected to follow a certain path through a strait should be given a series of way points (i.e., a *WayPoint* beliefset) so that they can make appropriate turns, rather than just maintain a course.

In comparison with the FIACs, the behaviours required for the neutral vessels should be fairly simple to create. With that said, there are many different types of neutral vessels that could be considered and each should exhibit unique behaviours. Once these neutral behaviours exist, it would be interesting to give the red force entities a plan to 'act like' a given neutral vessel. This behaviour could replace the 'act innocent' behaviour, and would potentially make the red entities more difficult to detect. Alternative types of red force entities should be considered as well, although it may be most effective to build these particular behaviours as they are specified and required by projects that the resulting scenarios and experimentation will support.

Subject Matter Experts (SMEs) should be used to validate the behaviours of all of the entities. To date, this project has focused on being able to take an idea of how an entity should act and making the agent software produce that behaviour. It has not focused on what the correct

behaviours really are (though they are based on red force behaviours from the VBE-E program). It is anticipated that agents will be used to control some (if not all) of the computer-controlled entities for the VCS group's experimentation in support of the Force Protection Technical Demonstration Program (TDP). Clearly some research into how the scenario should be set up and how it should play out will have to be completed, and then modifications or additions to the existing agents will be required. As time goes on, the library of agents and behaviours will be built up, and the amount of effort to create a new scenario will decrease.

The efficient development of scenarios, however, could go well beyond having many agents and behaviours to choose from. While this is the shorter term goal, the longer term desire is to allow scenario definition at a higher level. As in past experiments, for the neutral traffic in particular, the exact position or movement of an entity is not critical. The neutral entities need only 'act realistic'; specifying the finer details of each twist and turn for each neutral entity (such as is required with JSAF) is time consuming and unnecessary. It would be preferable for the scenario developer to specify things such as the location of fishing ports and fishing grounds (and maybe go as far as to specify the number of entities and the distribution of entity types), and then let the software figure out where to place the entities and how to move them. When behaviours for the neutral entities are built, this longer term goal should be kept in mind. For example, when fishing boats or lobster boats are built, they could include beliefsets that tell the entities where their home port is and where the fishing areas are, and they could have plans that move them from port to the fishing area, that move them around within the fishing area, and that move them back to their port (or possibly to an alternative port) at the end of the day. To take this a step further, the scenario developer could specify the time of day which could affect where the entities begin and which plan is active at the start of the simulation.

There are other issues that should be considered as well. These are not specific to agents, but rather are VMSA-wide issues that have not been addressed within the VMSA community. However, they are directly related to entity intelligence. The first is entity collision avoidance. VMSA entities will 'happily' travel through each other. This is an issue that could quite reasonably be handled by agents, assuming that a set of rules regarding who moves when and how could be obtained, and that all non-human controlled entities were controlled by the agents so that they were all following the same rules. The second, more complicated issue is that VMSA entities are not map-aware. When operating in open water, this is less of an issue. However, when operating in a strait or harbour, the entities need to be prevented from maneuvering onto land. For the initial set-up discussed in this paper, only the 'act innocent' behaviour was at risk of causing the entities to move onto land. To manage this, they were only allowed to 'act innocent' within a preset distance of their initial position. This was a quick fix, and not a long term solution. Some more thought is required to determine the appropriate way to provide entities with map-awareness. It should be noted that JSAF already handles both of these issues.

7 Concluding Remarks

This paper has shown that a real-world scenario can be modeled using JACK agent constructs such as beliefsets, events and plans. It has discussed the benefits of using agents to control computer generated forces in terms of realism, ease of scenario development, and repeatability. Future work in this area includes the development of additional entity types and behaviours as well as tools and processes to make scenario development more efficient. The true utility of agents for scenario development and entity control, however, will only be fully understood when they are used and applied to an experimental program and exercised in an experimental setting; above all, the operators must perceive the entities as believable.

References

[1] Canney, S.A., "Virtual Maritime System Architecture Description Document", Issue 2.00, Virtual Maritime System Document Number 00034, Defence Science and Technology Organisation, Edinburgh, Australia, July 2002.

[2] Hazen, M.G., Crebolder, J.M., Randall, T.E., Kisway, T., "VBE-E Experimentation", DRDC Atlantic Technical Memorandum TM 2006-309, Defence R&D Canada – Atlantic, Dartmouth, Nova Scotia, Canada, *in draft*.

[3] Randall, T.E., "An Introduction to Software Agents", DRDC Atlantic Technical Memorandum TM 2007-221, Defence R&D Canada – Atlantic, Dartmouth, Nova Scotia, Canada, *in draft*.

[4] "JACK Intelligent Agents: Agent Manual", Release 5.2, Agent Oriented Software Pty. Ltd., Carlton South, Victoria, Australia, June 2005.

[5] "JACK Intelligent Agents: Development Environment Manual", Release 5.2, Agent Oriented Software Pty. Ltd., Carlton South, Victoria, Australia, June 2005.

[6] Gaudet, B.J., "JACK/VMSA Bridge: User Guide and Technical Description", DRDC Atlantic Technical Memorandum, Defence R&D Canada – Atlantic, Dartmouth, Nova Scotia, Canada, *in draft*.

[7] Padgham, L. and Winikoff, M., "The Prometheus Methodology", RMIT University, Melbourne, Australia, April 2004, available at: http://goanna.cs.rmit.edu.au/~linpa/Papers/bookchB.pdf.

[8] Padgham, L. and Winikoff, M., "Prometheus: A Methodology for Developing Intelligent Agents", RMIT University, Melbourne, Australia, April 2004, available at: http://www.jamesodell.com/AOSE02-papers/aose02-04.pdf.

[9] Padgham, L. and Winikoff, M., "Developing Intelligent Agent Systems", John Wiley & Sons, Ltd., 2004.

[10] Randall, T.E. and Gaudet, B.J., "Implementing Priority Task Management in JACK Intelligent Agents", to be published in Proceedings of SpringSim'08, 2008.

Distribution list

Document No.: DRDC Atlantic TM 2007-342

LIST PART 1: Internal Distribution by Centre:

5 DRDC Atlantic Library
1 Tania E. Randall
1 Mark G. Hazen
1 H/MICS
1 Allan Gillis
1 Jason Murphy
1 Brad Dillman
1 Glenn Franck

12 TOTAL LIST PART 1

LIST PART 2: External Distribution by DRDKIM

2 DRDKIM
1 H/SECO, CFMWC, PO Box 99000, Halifax, NS B3K 5X5
4 DRDC Valcartier, 2459 Boul. Pie XI Nord, Val-Bélair, QC G3J 1X5
Abder Benaskeur
Éloi Bossé,
François Bernier
Marielle Mokhtari
1 DSTM, NDHQ, 101 Colonel By Drive, Ottawa, ON K1A 0K2
1 UK National Leader MAR TP-1, Robert Walden, Defence Science & Technology Laboratory, C134/NDS, Dstl Portsdown West, Portsdown Hill Road, Fareham Hampshire, UK P017 6AD
1 New Zealand National Leader MAR TP-1, Defence Technology Agency, Private Bag 32901, Auckland Naval Base, Auckland, NZ
1 Australia National Leader MAR TP-1, Mark Kreig, Defence Science & Technology Organization, Maritime Operations Division, Edinburgh, South Australia 5111
1 US National Leader MAR TP-1, Erik Chaum, Code 2231, Naval Undersea Warfare Center, 1176 Howell Street, Newport, RI, 02841-1708.

12 TOTAL LIST PART 2

24 TOTAL COPIES REQUIRED

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence R&D Canada – Atlantic
9 Grove Street
P.O. Box 1012
Dartmouth, Nova Scotia B2Y 3Z7

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C, R or U) in parentheses after the title.)

Using Software Agents to Control the Behaviour of Simulated Entities

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Randall, T.E.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
December 2007	41	10

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Atlantic
9 Grove Street
P.O. Box 1012
Dartmouth, Nova Scotia B2Y 3Z7

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
11bt	

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Atlantic TM 2007-342	

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

The Virtual Combat Systems (VCS) group at Defence R&D Canada – Atlantic (DRDC Atlantic) has been developing a virtual maritime environment for command and control (C2) level, human-in-the-loop experimentation using the High Level Architecture (HLA)-based Virtual Maritime Systems Architecture (VMSA). For typical experiments using VMSA, human operators have been used to control the friendly (blue) force entities during an experiment while the opposing (red) force entities have been controlled by the simulation. Most recently, the red force entities have been modelled within the Joint Semi-Automated Forces (JSAF) environment. While the sensor and weapon models inside of JSAF are believed to be of a good fidelity, the ability to describe an entity's behaviour and decision making is lacking. In addition, JSAF scenarios are tedious to develop and require human input at run-time, creating problems in terms of scenario repeatability.

In this paper, the use of agent-based technology is investigated as a method for improving the definition, autonomy and realism of red force entities for use in VMSA simulations. The JACK Intelligent Agent software was selected for developing behaviours and decision making capabilities for simulation-controlled entities. In the application discussed in this document, JACK is used to decide when an entity should move, share information with another entity, or fire its weapons, while VMSA is used to model the physical aspects of the entities, such as motion, sensors and weapons.

A scenario involving a group of fast inshore attack craft (FIACs) claiming control over a narrow strait is modelled. The paper focuses on the process of taking a conceptual description of this scenario and mapping it into agent constructs (events, goals, plans, and beliefs) that are meaningful to the JACK software. It concludes that agent technology *can* be used to effectively control computer generated forces in a VMSA simulation and discusses the benefits of this approach.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

JACK Intelligent Agents, High Level Architecture (HLA), virtual environment, VMSA, behaviour representation

This page intentionally left blank.

Defence R&D Canada

Canada's leader in defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

www.drdc-rddc.gc.ca