

# Using Agent-based Simulation in a Decision Support System for Military Command & Control

T. T. Luik BSc.  
Faculteit Exacte Wetenschappen,  
Vrije Universiteit Amsterdam

August 6, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF  
MASTER OF SCIENCE  
IN  
ARTIFICIAL INTELLIGENCE

Project Supervisors

Dr. T. Bosse (VU Amsterdam)

Ir. N.M. de Reus (TNO Den Haag)



Project Period

September 2011 - August 2012

## **Abstract**

In the military, command & control (C2) systems are used by the commanders to provide orders to their troops and to monitor the execution of these orders. By connecting such systems to a simulator, the simulator's computer generated forces can replace actual human forces, providing cheap, quick and reusable units for purposes such as training, mission preparation or even support during mission execution. To allow this connection, TNO created a multi-agent system to understand the orders from the military commander and subsequently provide the simulator with the correct tasks to execute the given order. Building on this foundation, this Master's project's task was to take it one step further by finding and creating a prototype application for these agents. We chose to apply our multi-agent system in a decision support system (DSS). For a decision support system based on simulation to actually provide support to the commander, its interface needs to be usable in the C2 process and the simulation's results need to be realistic. To solve these problems, requirements from related research and military experts on both these matters were consulted. Based on these examples, our decision support system now provides a table in which multiple orders from the C2 system can be mapped to an analysis of their execution in the simulator. During, prior or after simulation, these analysis criteria can be shown or hidden, allowing the user to compare the orders on only the most interesting criteria. To increase the realism of the simulation's results, military doctrine has been introduced to the multi-agent system. Now the agents can break up an assault in the required phases, move in the correct formations and respond to unexpected enemy behaviour with a blocking position, based on doctrinal documents provided by the Royal Netherlands Army. By doing this, we have shown a glimpse of the future where the computer generated forces realistically mimic soldiers' behaviour like teamwork, coordination, reactive capabilities and the deliberate doctrinal planning, thereby allowing the simulator to realistically foresee the results of the commander's orders. When used in conjunction with an expanded version of our decision support system, commanders will be able to realistically evaluate and compare their courses of action and choose the best one for accomplishing their current mission.

## **Acknowledgements**

I would like to thank Tibor Bosse for supervising my Master's project from the VU, providing support and guidance on writing this thesis and on the grand layout of the project. Next, I would like to thank Nico de Reus for providing me this opportunity to do my project at TNO and for guiding me on a weekly basis, for arranging meetings with the experts and much more. Also, I would like to thank Henk Henderson for his support on the programming side of this project. And finally, I would like to thank the rest of the TNO Modelling, Simulation and Gaming group for showing me a good time at TNO; and for allowing me to join in the personal-growth sessions with Dutch guru Remco Claassen!



# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                                     | <b>8</b>  |
| 1.1      | Problem Domain . . . . .                                | 9         |
| 1.1.1    | TNO . . . . .   | 9         |
| 1.1.2    | Command Agents . . . . .                                | 10        |
| 1.2      | Solution Range . . . . .                                | 10        |
| 1.3      | Research Question . . . . .                             | 11        |
| 1.3.1    | Usability . . . . .                                     | 11        |
| 1.3.2    | Realism . . . . .                                       | 11        |
| 1.4      | Thesis Outline . . . . .                                | 12        |
| <b>2</b> | <b>Background and Related Work</b>                      | <b>13</b> |
| 2.1      | Command & Control (C2) . . . . .                        | 14        |
| 2.1.1    | Command Hierarchy . . . . .                             | 14        |
| 2.1.2    | Military Decision-Making Process (MDMP) . . . . .       | 15        |
| 2.1.3    | Course of Action (COA) . . . . .                        | 15        |
| 2.1.4    | Integrated Staff Information System (ISIS) . . . . .    | 16        |
| 2.2      | Simulation Technologies . . . . .                       | 16        |
| 2.2.1    | Simulation Standards . . . . .                          | 16        |
| 2.2.1.1  | Coalition Battle Management Language (C-BML) . . . . .  | 17        |
| 2.2.1.2  | Military Scenario Definition Language (MSDL) . . . . .  | 17        |
| 2.2.1.3  | High-Level Architecture (HLA) . . . . .                 | 17        |
| 2.2.2    | Computer Generated Forces (CGF) . . . . .               | 18        |
| 2.2.2.1  | CGF Requirements . . . . .                              | 19        |
| 2.2.2.2  | VR-Forces . . . . .                                     | 20        |
| 2.2.3    | Agents . . . . .  | 21        |
| 2.2.3.1  | Belief-Desire-Intention (BDI) model . . . . .           | 22        |
| 2.2.4    | TNO Architectures . . . . .                             | 24        |
| 2.2.4.1  | System Architecture . . . . .                           | 24        |
| 2.2.4.2  | Agent Architecture . . . . .                            | 24        |
| 2.3      | Decision Support System for Command & Control . . . . . | 26        |
| 2.3.1    | Previous C2 DSS projects in literature . . . . .        | 26        |
| 2.3.2    | DSS Requirements . . . . .                              | 28        |

|          |   |           |
|----------|---|-----------|
| <b>3</b> | <b>Methods</b>  | <b>31</b> |
| 3.1      | Expert Information Solicitation . . . . .                           | 32        |
| 3.2      | Hardware & Software . . . . .                                       | 33        |
| 3.3      | Validation . . . . .  | 33        |
| <b>4</b> | <b>Model</b>  | <b>34</b> |
| 4.1      | Increasing Realism . . . . .  | 35        |
| 4.1.1    | Agent Behaviour . . . . .   | 35        |
| 4.1.2    | Combat Instruction Sets (CISs) . . . . .                            | 36        |
| 4.1.2.1  | Scenario . . . . .  | 36        |
| 4.1.2.2  | Assault . . . . .   | 37        |
| 4.1.2.3  | March and Formations . . . . .                                      | 39        |
| 4.1.2.4  | Blocking Position . . . . .   | 43        |
| 4.2      | System Architecture . . . . .                                       | 44        |
| 4.3      | Conceptual Design - Agent . . . . .                                 | 44        |
| 4.3.1    | Agent Architecture . . . . .  | 45        |
| 4.3.2    | HLA Manager . . . . .   | 45        |
| 4.3.3    | Information Manager . . . . .                                       | 46        |
| 4.3.4    | Stub agent . . . . .  | 46        |
| 4.3.5    | Battalion agent . . . . .   | 46        |
| 4.3.6    | Company agent . . . . .   | 46        |
| 4.3.7    | Platoon agent . . . . .   | 47        |
| 4.4      | Conceptual Design - Decision Support System (DSS) . . . . .         | 47        |
| 4.4.1    | Initial concept . . . . .   | 47        |
| 4.4.2    | Measures of Effectiveness (MOEs) . . . . .                          | 48        |
| <b>5</b> | <b>Implementation</b>   | <b>50</b> |
| 5.1      | Communicating to VR-Forces: New in- & output requirements . . . . . | 51        |
| 5.1.1    | Set Heading, Turn To Heading & Set Speed . . . . .                  | 52        |
| 5.1.2    | Abort executing task . . . . .                                      | 54        |
| 5.1.3    | Move into formation . . . . .                                       | 54        |
| 5.1.4    | Follow . . . . .  | 56        |
| 5.1.5    | EntityFuel & EntityAmmunition . . . . .                             | 56        |
| 5.2      | Communicating with C2IS: Processing the MSDL ORBAT . . . . .        | 58        |
| 5.3      | Agent Behaviour: Implementation of the CISs . . . . .               | 59        |
| 5.3.1    | CIS: Assault on Enemy Position . . . . .                            | 59        |
| 5.3.1.1  | Phase 1 : Preparation . . . . .                                     | 59        |
| 5.3.1.2  | Phase 2 : Fire and Movement . . . . .                               | 61        |
| 5.3.1.3  | Phase 3 : Final attack . . . . .                                    | 61        |
| 5.3.1.4  | Phase 4 : Consolidate and Reorganize . . . . .                      | 61        |
| 5.3.1.5  | Finalizing the assault . . . . .                                    | 61        |
| 5.3.2    | CIS: Tactical Road March . . . . .                                  | 62        |
| 5.3.2.1  | TacticalRoadMarch plan . . . . .                                    | 62        |
| 5.3.2.2  | OffRoadMarch plan . . . . .   | 63        |

|          |  |           |
|----------|--|-----------|
| 5.3.3    | CIS: Hasty Occupation of a Blocking Position . . . . . | 63        |
| 5.3.3.1  | Distance threshold . . . . .                           | 63        |
| 5.3.3.2  | Expected locations . . . . .                           | 63        |
| 5.3.3.3  | Blocking positions . . . . .                           | 63        |
| 5.3.3.4  | Suspension of other tasks . . . . .                    | 64        |
| 5.3.3.5  | Ending the blocking positions . . . . .                | 65        |
| 5.3.3.6  | Diagrams . . . . .                                     | 65        |
| 5.3.4    | CIS formations . . . . .                               | 65        |
| 5.3.4.1  | Positional offset . . . . .                            | 67        |
| 5.3.4.2  | Heading . . . . .                                      | 69        |
| 5.3.4.3  | CIS: Column Formation (Tank Company) . . . . .         | 69        |
| 5.3.4.4  | CIS: Column Formation (Tank Platoon) . . . . .         | 69        |
| 5.3.4.5  | CIS: Line Formation Traveling (Tank Company) . . . . . | 70        |
| 5.3.4.6  | CIS: Line Formation Traveling (Tank Platoon) . . . . . | 70        |
| 5.4      | DSS . . . . .  | 71        |
| 5.4.1    | Implementing the GUI . . . . .                         | 71        |
| 5.4.1.1  | Table . . . . .  | 71        |
| 5.4.1.2  | Listening to the DSS . . . . .                         | 71        |
| 5.4.2    | Computing the MOEs . . . . .                           | 72        |
| 5.4.2.1  | Survival rate . . . . .                                | 72        |
| 5.4.2.2  | Fuel percentage & ammunition percentage . . . . .      | 73        |
| 5.4.2.3  | Completed . . . . .                                    | 73        |
| 5.4.2.4  | Enemy survival rate . . . . .                          | 74        |
| <b>6</b> | <b>Results</b>   | <b>75</b> |
| 6.1      | DSS Behaviour . . . . .                                | 76        |
| 6.1.1    | Connecting DSS with an order . . . . .                 | 77        |
| 6.1.2    | Hiding MOEs . . . . .                                  | 77        |
| 6.1.3    | Retrieving MOEs for the order . . . . .                | 78        |
| 6.1.4    | Comparing MOEs . . . . .                               | 79        |
| 6.2      | CGF Behaviour . . . . .                                | 80        |
| 6.2.1    | Marching . . . . .                                     | 84        |
| 6.2.1.1  | Previous Behaviour - Move . . . . .                    | 84        |
| 6.2.1.2  | Tactical Road March . . . . .                          | 84        |
| 6.2.1.3  | Column Formation . . . . .                             | 84        |
| 6.2.1.4  | Line Formation . . . . .                               | 86        |
| 6.2.2    | Assault on a enemy position . . . . .                  | 86        |
| 6.2.2.1  | Previous Behaviour - Seize . . . . .                   | 86        |
| 6.2.2.2  | Phase 1: Preparation . . . . .                         | 87        |
| 6.2.2.3  | Phase 2: Fire and Movement . . . . .                   | 88        |
| 6.2.2.4  | Phase 3: Final Attack . . . . .                        | 88        |
| 6.2.2.5  | Phase 4: Consolidation . . . . .                       | 89        |
| 6.2.3    | Blocking Position . . . . .                            | 90        |

|          |  |            |
|----------|--|------------|
| 6.2.3.1  | Enemy Presence . . . . .   | 90         |
| 6.2.3.2  | Blocking Position occupied . . . . .                                     | 91         |
| 6.2.3.3  | Enemy destroyed . . . . .  | 91         |
| 6.2.3.4  | Blocking Position completed . . . . .                                    | 93         |
| <b>7</b> | <b>Discussion</b>  | <b>94</b>  |
| 7.1      | Usability . . . . .  | 95         |
| 7.1.1    | Expert desires . . . . .   | 95         |
| 7.1.2    | Literature requirements . . . . .  | 96         |
| 7.2      | Realism . . . . .  | 99         |
| 7.2.1    | Level of realism . . . . .   | 99         |
| 7.2.2    | Requirements . . . . .   | 100        |
| 7.2.2.1  | Architecture . . . . .   | 100        |
| 7.2.2.2  | Autonomous Operation . . . . .   | 100        |
| 7.2.2.3  | Realistic Behaviour . . . . .  | 100        |
| 7.2.2.4  | Organization . . . . .   | 101        |
| 7.3      | BDI agents for C2-Simulation interoperability - good approach? . . . . . | 102        |
| 7.4      | Future Work . . . . .  | 103        |
| 7.4.1    | Remark about level of realism and discussion with RNLA experts . . . . . | 103        |
| 7.4.2    | DSS . . . . .  | 104        |
| 7.4.3    | CGF requirements . . . . .   | 105        |
| <b>8</b> | <b>Conclusions</b>   | <b>106</b> |
|          | <b>Bibliography</b>  | <b>110</b> |
| <b>A</b> | <b>AI Upgrade Possibilities</b>  | <b>114</b> |
| A.1      | Agent Behaviour . . . . .  | 115        |
| A.2      | Path Planning . . . . .  | 116        |
| A.3      | Enemy Behaviour . . . . .  | 116        |
| <b>B</b> | <b>Combat Instruction Sets (CISs)</b>                                    | <b>118</b> |
| <b>C</b> | <b>UML Diagrams</b>  | <b>120</b> |
| C.1      | Order processing . . . . .   | 121        |
| C.2      | UML Activity Diagram - CIS Blocking Position . . . . .                   | 121        |
| C.2.1    | Unit Detection Update diagram . . . . .                                  | 122        |
| C.2.2    | Blocking Position Battalion diagram . . . . .                            | 122        |
| <b>D</b> | <b>Military Scenario Definition Language (MSDL)</b>                      | <b>125</b> |
| D.1      | MSDL elements . . . . .  | 126        |
| D.1.1    | msdl:MilitaryScenario Element . . . . .                                  | 126        |
| D.1.2    | msdl:Organizations . . . . .   | 126        |
| D.1.3    | msdl:Units . . . . .   | 126        |
| D.1.4    | msdl:Unit . . . . .  | 126        |

|          |                              |            |
|----------|------------------------------|------------|
| D.1.5    | msdl:Equipment . . . . .     | 126        |
| D.1.6    | msdl:EquipmentItem . . . . . | 126        |
| <b>E</b> | <b>FOM</b>                   | <b>132</b> |
| E.1      | FOM changes . . . . .        | 133        |

# Glossary

B-HAVE AI navigation module in VR-Forces

BDI Belief-Desire-Intention agent paradigm

C-BML Coalition Battle Management Language

C2 Command and Control

C2IS Command and Control information system

C2WS Command and Control workstation, a synonym for C2IS

CGF Computer Generated Forces

CIS Combat Instruction Set

COA Course of Action

COP Common Operational Picture

COTS Commercial Off-The-Shelf

CxBR Context-based Reasoning agent paradigm

DSS Decision Support System

FFI Norwegian research institute Forsvarets forskningsinstitutt

FOM Federation Object Model

GIS Geographic Information System

GUI Graphical User Interface

HLA High-Level Architecture

ISIS RNLA's C2IS called Integrated Staff Information System

Jadex Reasoning engine for BDI agents

Low-level BML Low-level Battle Management Language, TNO and FFI's extensions to C-BML

M1A2 M1 Abrams tank

MAS Multi-Agent System

MDMP Military Decision-Making Process

MOE Measure of Effectiveness

MSDL Military Scenario Definition Language

MSG Modelling, Simulation and Gaming department of TNO in the Hague

MSG-048 NATO's Modelling and Simulation Group 048

MSG-085 NATO's Modelling and Simulation Group 085

NATO North Atlantic Treaty Organisation

NPC Non-Player Character

OpOrder Operation Order Template

ORBAT Order of Battle

RNLA Royal Netherlands Army

ROE Rules of Engagement

RPR FOM Real-time Platform Reference FOM, SISO's FOM standard

RTS Real-Time Strategy

SISO Simulation Interoperability Standards Organization

TNO Netherlands Organisation for Applied Scientific Research

UML Unified Modeling Language

VR-Forces Virtual Reality Forces, a simulator from MÄK Technologies

VU Vrije Universiteit Amsterdam

XML Extensible Markup Language

## Chapter 1

### Introduction



## 1.1 Problem Domain

Igitur qui desiderat pacem, preparet bellum.

Vegetius Renatus (390)

Vegetius' (~390 AD) quote can be translated as: "Therefore whoever desires peace, let him prepare for war". Still in this day and age it remains current, as nations around the world continue to try and obtain a competitive edge with new military technology. With the advent of the computer, military forces have created command & control (C2) systems to support the commanding staff in controlling their military operations. While the soldiers are executing an operation on the battlefield, the commanding staff resides off the battlefield with the C2 information system (C2IS<sup>1</sup>), trying to gain situational awareness of the current state of the operation, so that the best informed decisions can be made. A recent development in this area is the interoperability between C2 systems and simulation. Instead of providing the C2 system with real-time input from sensors, simulation systems can be plugged in to model battlefield situations and simulate this input. If successfully integrated, staff can be trained in a cost-effective way using the preferred *train as you fight* paradigm. Moreover, realistic battlefield situations can be simulated before or parallel to an operation, to support the commanding staff in their decision making before or during an operation. As such, C2 - simulation interoperability should be of great benefit to whoever desires peace.

### 1.1.1 TNO

The research for this Master's thesis has been conducted at TNO, an independent Dutch research institute. TNO is contracted by, among others, the Netherlands' Ministry of Defence to perform their defence research. Furthermore, TNO, as Dutch representative, has been part of multiple NATO Modelling and Simulation task groups. This thesis is motivated partly by the work TNO is doing for one of these task groups: MSG-085<sup>2</sup>. MSG-085 is responsible for evaluating the Coalition Battle Management Language (C-BML), which is a standardized language for communication of orders between military C2 systems and simulation systems (Bronkers et al., 2011). For a previous NATO task group, MSG-048<sup>2</sup>, TNO and FFI (Norway<sup>3</sup>) have co-developed a C-BML interface for their C2 systems. This interface allows the C2 systems to communicate using C-BML. However, current simulators can not directly process C-BML orders from the C2 systems yet. While C2 systems issue high-level orders, simulators act on low-level orders, e.g. they need an order for single units instead of groups. To bridge this gap, both research institutes are developing a multi-agent system (MAS) to translate the orders from high-level to low-level. This MAS has been modelled by the two countries using two different agent models: TNO is using the Belief-Desire-Intention (BDI) paradigm, while FFI uses the Context-Based Reasoning (CxBR) paradigm. In the future, a comparison between the two different approaches is to be made (Bronkers et al., 2011).

---

<sup>1</sup>[http://www.defensie.nl/english/army/materiel/communication\\_and\\_information\\_systems/information\\_systems/command\\_control\\_workstation](http://www.defensie.nl/english/army/materiel/communication_and_information_systems/information_systems/command_control_workstation)

<sup>2</sup><http://www.cso.nato.int/activities.aspx>

<sup>3</sup><http://www.ffi.no/en/Sider/default.aspx>

### 1.1.2 Command Agents

The TNO MAS, which is named *Command Agents* by Bronkers (2011), has to replace part of the military command hierarchy to transform the high-level orders. For this reason, a similar hierarchy of agents has been implemented: the Battalion agent, the Company agent and the Platoon agent. This should allow for a realistic simulation of how an order actually traverses down the command hierarchy to the soldiers in the field. Moreover, expert knowledge used by these command echelons can be transferred to their respective agent.

The Command Agents can let the simulator perform three simplified orders: *Attack*, *Defend* and *Seize*. Here *Attack* is a task where a position is given, towards which the units should travel, and the unit's rules of engagement (ROE) are set to *fire at will*. The simulation then concludes from these ROE that the unit should fire upon all enemies in sight. *Defend* is similar to *Attack*, but the ROE are set to *fire when fired upon*. *Seize* is a combination of these two: first the *Attack* order is given, and after completion of this *Attack*, the *Defend* order is given for the same area.

However, the agents do not yet exhibit a large amount of autonomy, intelligence or realistic behaviour. They respond to the commands provided by the user by channelling simplified versions through to the simulator. While already admirable, there are a lot of situations on the simulated battlefield where autonomous tactical decision making is required. In further development of the Command Agents, we want more of this reactive and proactive behaviour.

## 1.2 Solution Range

Recently, TNO started looking for an application of their Command Agents. This application in the C2 domain could be either in *training* or in *decision support*.

In military training, simulation is already a heavily used alternative to the expensive live training. To allow the commanders to train as they fight, a multitude of human operators is needed behind the scenes to translate the commands given by the trainee to orders in the simulator and to report the results back. The Command Agents could automate part of the activities of these human operators, cutting on costs and making it easier to initiate these training sessions.

As military decision support, simulation could be used to compare multiple plans by providing the users with repeatable, objective, evidence-based results on the wargaming situation. Currently, military plans (courses of action or COAs) are tested subjectively by the commander and his staff, or not tested at all. Using a simulation decision support system might also allow usage during an operation, where new events on the battlefield could quickly be analyzed in the simulator to see if the current COA is still feasible.

For this project, we have opted for identifying a simple decision support case to show the potential of the Command Agents. This decision was made because the scope can be smaller and thus results can be obtained faster in a decision support application than in a training application.

In interviews with military experts, and in a review of related literature, requirements for this case will be acquired. According to these requirements, a decision support system (DSS)

will be built that uses the agent-powered simulation to provide some relevant results to its users. Furthermore, increased realism will be required from the Command Agents to make these results more useful. The agents replace command echelons where a lot of tactical reasoning is employed to specify the received command to lower-level units. This reasoning should now be executed by the agents, while the simulator only takes care of replacing the individual soldier's or vehicle's behaviour on the battlefield. Adding increasingly realistic behaviour to the Command Agents is also part of this Master's project.

### 1.3 Research Question

Our main research question is:

"How can we create a useful decision support system for military planning, using the combination of simulation and C2?"

To create a useful decision support system (DSS) using simulation, the system needs to be usable for military commanders and the results provided need to be realistic. Thus, the multiple subquestions, entailed in our main research question, are divided into two categories:

- Usability
- Realism

#### 1.3.1 Usability

The usability problem is present in every decision support system (DSS): if the user is to be supported, the system has to fit in with the user's actions and environment. The benefits of using the DSS should outweigh its costs, otherwise the system actually does not support decision making. Therefore, we will answer: how can we make a usable DSS in the C2 domain?

To this end, we will figure out requirements of a DSS for C2 from related work on this matter. Furthermore, we will pick a part of military planning from C2 that is applicable for DSS support. This allows questions such as what DSSs have previously been used in this area of C2? Finally we will figure out from these requirements, related work and expert information, what the user interface of the DSS should be like. This entails how results from our simulation will be represented by the DSS. For interoperability, the DSS will need to interact with the simulator and the agents, so we will find a way to connect it to the current system framework.

#### 1.3.2 Realism

Even if a DSS is usable, it can only be useful if it has some substance, if the information it provides is realistic. Realism of the DSS is based on the results from the simulation. For this reason, we also need to answer the following question in our project: how can we make the simulation results more realistic?

The factors influencing the simulator's realism are its own models, the agents providing the orders and the context or scenario created during setup. The first depends on the simulator used, but the latter two are modifiable.

We have identified three distinct ways of showing realism in the agent: response to orders, environment and enemy actions. More realistic response to orders might include comprehension of more than three orders or allowing more detailed orders. For example, allow one order that is conditional on the success of another. More realistic response to the environment might include more realistic path planning, where the agent could take into account the type of roads, visibility, expected danger, and more. More realistic response to enemy actions concerns itself with the dynamic changes in the environment caused by opposing forces. If fired upon, the agent might want to adjust its current plan, starting with a hasty retreat.

Finally, a more realistic context can be created by using e.g. realistic names, command hierarchies and unit types. This should allow for a quicker comprehension of results by the military user.

## 1.4 Thesis Outline

In Chapter 2 we will discuss the background information needed to understand what is written. Then Chapter 3 will explain the methods we used, Chapter 4 will provide the model we have created and Chapter 5 will describe how we implemented this model. This setup will of course be followed by Chapter 6, in which we will show the results we have obtained. Finally, a discussion of the results can be found in Chapter 7 and our conclusions in our final Chapter 8.

## Chapter 2

# Background and Related Work

In this chapter, we shall provide extra information on the subjects needed to comprehend the rest of this thesis. The outline is as follows: In section 2.1, information about military command & control principles is provided. Then in section 2.2, we discuss the simulation technologies and agents used in this project. Finally, in Section 2.3, decision support system requirements are discussed.

## 2.1 Command & Control (C2)

”The exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of the mission. Command and control functions are performed through an arrangement of personnel, equipment, communications, facilities, and procedures employed by a commander in planning, directing, coordinating, and controlling forces and operations in the accomplishment of the mission. Also called C2.”

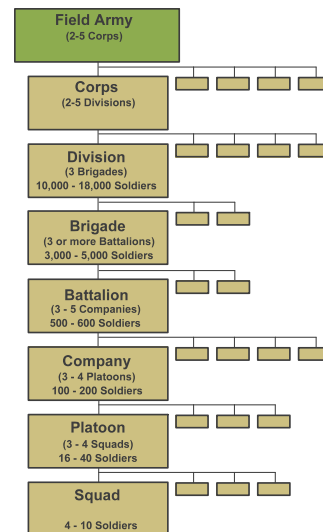
(Joint Education and Doctrine Division, J-7, Joint Staff, U.S. Department of Defense, 2011)

Military command & control contains the planning of the operation and the coordination of the troops by the military commanders. It is a concept of centralized decision making, where a commanding staff needs to manage a diverse collection of units and equipment in order to obtain a particular effect, usually against the will of an opposing force.

### 2.1.1 Command Hierarchy

A modern army is divided into multiple levels of command (called echelons), creating a command hierarchy. For example, look at such a command hierarchy for the U.S. Army in Figure 2.1. Note that the Royal Netherlands Army (RNLA), our target audience, can only field 3 Brigades (Royal Netherlands Army, Ministry of Defence, 2011), so its hierarchy is a subsection of Figure 2.1. In the previous work done by TNO (Bronkers, 2011), the command hierarchy has been limited to the battalion, company and platoon levels. Instead of the squad level, individual tanks were used (i.e. each platoon consisted of 4 tanks). As the RNLA has disbanded all cavalry due to budget cuts<sup>1</sup>, other vehicles

might be simulated. However, the actual soldiers and equipment (including vehicles) that make up these echelons in a particular situation are given in an order of battle (ORBAT). For

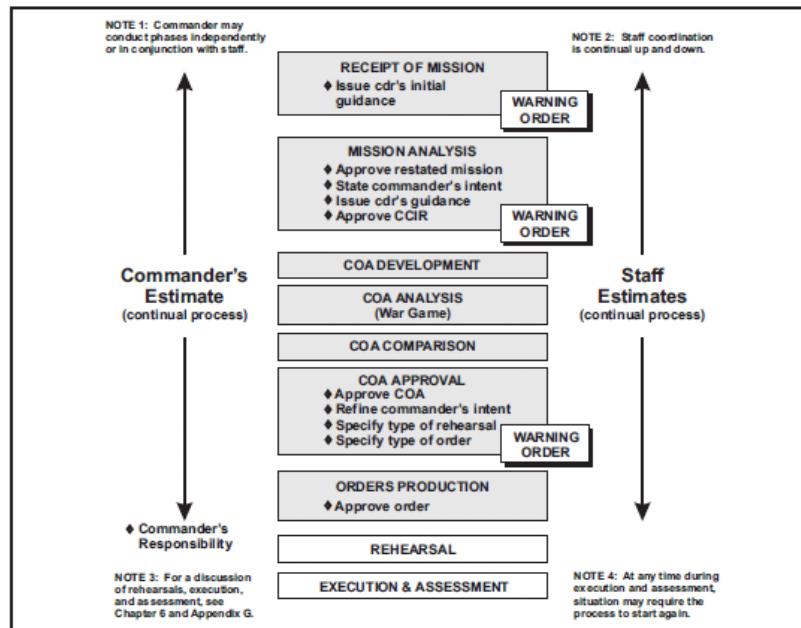


**Figure 2.1:** *The Operational Unit Diagram from U.S. Army (2011)*

<sup>1</sup>[http://www.defensie.nl/english/latest/news/2011/05/26/48183133/Dutch\\_tank\\_history\\_ends\\_with\\_a\\_bang](http://www.defensie.nl/english/latest/news/2011/05/26/48183133/Dutch_tank_history_ends_with_a_bang)

an increasingly realistic context, automatically connecting the ORBAT from the C2 system to the simulator shall be considered in this project. This would allow the Command Agents to automatically create the correct agents for the current scenario and the simulator to simulate the correct equipment.

### 2.1.2 Military Decision-Making Process (MDMP)

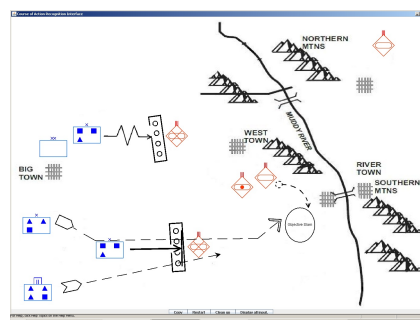


**Figure 2.2:** *The Military Decision-Making Process, see Figure 5-1 in FM 101-5 Staff Organization and Operations (1997)*

The MDMP is a planning method for military decision-making. It shows on a high level how the military command staff should plan their operation. Its steps are receipt of mission, mission analysis, course of action (COA) development, COA analysis (wargaming), COA comparison, COA approval and orders production. Since this is the method used in C2 decision making, our DSS should naturally fit into one (or more) of these levels before it can be useable. The middle three COA steps are where the plan for a mission is made and thus these are most suitable for decision support using simulation.

### 2.1.3 Course of Action (COA)

A course of action (COA) is a combination of orders for different military units that should lead to the accomplishment of the mission. Its



**Figure 2.3:** *A Course of Analysis sketch (Stolt, 2007)*

creation is a part of the MDMP. As can be seen in the example COA in Figure 2.3, a COA might be sketched on an abstract (digital) map using high-level units. The commanding staff of the higher echelons only concern themselves with the grand layout of the operation (e.g. company level). These commands are propagated downwards to the soldiers on the field by the commanders of the lower echelons in the hierarchy, as seen in section 2.1.1.

#### 2.1.4 Integrated Staff Information System (ISIS)

ISIS is a command & control information system (C2IS), used by the RNLA and TNO, providing the commanding staff with an interactive environment to exercise their command and control. According to RNLA (2011), ISIS consists of three systems: Geographic Information System (GIS), Order of Battle (ORBAT) and Operational Order Template (OpOrder). GIS shows a digital map containing the Common Operational Picture (COP), e.g. real-time updated positions of friendly and enemy units, towns, buildings and bridges, shared by all command levels using this system. The ORBAT allows the commanding staff to input a military hierarchy in the system (e.g. which units make up which platoons, companies and battalions). OpOrder allows the commander to formulate its orders in a uniform manner using a Word template provided by the NATO. In the current setup, orders are provided by an overlay on GIS and then translated to C-BML (see Section 2.2.1.1) when sent to the agents.



**Figure 2.4:** *Military personnel using ISIS (RNLA, 2011)*

## 2.2 Simulation Technologies

In this section we will discuss information related to the simulation side of this project. This includes CGF, agents, C2-simulation interoperability standards and the architectures from Bronkers (2011).

### 2.2.1 Simulation Standards

Standards allow for reuse and interoperability of software components across multiple projects and people. Therefore, this project's architecture also uses standards from the simulation field. In fact, the NATO MSG-085 project that TNO is working on is part of evaluating the C-BML standard. The main maintainer and updater of simulation standards is Simulation Interoperability Standards Organization<sup>2</sup> (SISO), which is responsible for most of the following

---

<sup>2</sup><http://www.sisostds.org/>



standards.

Below I shall describe some of the standards that are used in our architecture: C-BML for the orders sent from a C2 information system (C2IS) to the agents; MSDL for the setup of the scenario (units, terrain, et cetera) in the simulator; and HLA as the transport bus and means of communication between the C2IS, the agents and the simulator.

#### 2.2.1.1 Coalition Battle Management Language (C-BML)

C-BML is a language standardized and maintained by SISO to be able to describe orders given in a COA, so that the orders provided in the C-BML language are universally understood by all simulators. When completed, C-BML should allow the tactical ideas in the head of the commander to be transferred to a computer-understandable format, without loss of information or ambiguity. The orders given in ISIS are translated to C-BML, in an XML format, and transferred to the agents. Reports generated by VR-Forces and the agents are also translated to C-BML before they are sent back to ISIS. In a future best-case scenario agents would not be needed as an intermediary, as the simulator would understand and comply with the C-BML language, executing orders as they are given by a C2IS.

Communication of orders and reports between the agent and the simulator is not part of a standard, and have been called low-level BML, as opposed to the (high-level) C-BML standard.

#### 2.2.1.2 Military Scenario Definition Language (MSDL)

MSDL is a language standardized and maintained by SISO, used to enable description of, among others, the ORBAT in a uniform way. This way all simulators should be able to understand the provided ORBAT when it is written using MSDL. MSDL can then be used to automatically synchronize the setup (number, type, etc.) of units in VR-Forces and the C2 system ISIS. A MSDL file is written in XML (Extensible Markup Language), and a XML schema<sup>3</sup> has been provided by SISO that describes the elements that should be adhered to. In Appendix D you can find examples of MSDL used during this project and descriptions of some elements of the MSDL XML schema.

Currently in use is the ORBAT provided in the MSDL file: military echelons are provided as Units, including their echelon level, their name, their superiors, et cetera. Other units are provided as Equipment, e.g. a tank with its crew, or an individual soldier.

#### 2.2.1.3 High-Level Architecture (HLA)

HLA is the High-Level Architecture that, when adhered to by all systems, provides interoperability between these systems. This means that the simulators will be able to share data effectively and interpret data from other simulators effectively (Dahmann et al., 1997). It is used as a communication line for common understandability in this project's system architecture, connecting ISIS (gateway), the agents and VR-Forces, transferring the C-BML and low-level-BML messages. These include all the updates about entities in the simulator and the orders provided from the C2 system.

---

<sup>3</sup>[http://www.sisostds.org/DigitalLibrary.aspx?Command=Core\\_Download&EntryId=30830](http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=30830)

**FOM** Specifications of what can be transferred via HLA are recorded in an XML file, referred to as the Federation Object Model (FOM). It is the ontology shared between the different simulators to understand what is received from and sent to HLA. When additional information is required from a simulator, the FOM will have to be adjusted so that other users of the HLA bus will be able to receive this information.

There is a FOM standardized by SISO since 1999 called the real-time platform reference FOM or RPR FOM, providing a common ontology for simulators. The FOM used in this project is the RPR FOM extended with low-level BML. More detailed description of (adaptions to) the FOM in our project will be provided in Chapter 4, Chapter 5 and Appendix E.

**HLA Evolved** During this project, an update of HLA, called HLA Evolved, has been taken in use. This newer version of HLA allows for modular FOMs, allowing for example the separation of the RPR FOM and our low-level BML extensions into two modules. When working properly, this should allow communication between simulators while they do not share the exact same FOM, as all FOM modules are added to the current HLA ontology. An example of its usefulness is that our simulator and agent, working with our extended FOM, can still communicate with a simulator from a coalition partner who, for example, only employs the RPR FOM.

### 2.2.2 Computer Generated Forces (CGF)

In this subsection, we will present the basics about computer generated forces (CGF). A description of this term, given by M & S Office, U.S. Army (2011), is provided below.

“A generic term used to refer to computer representations of entities in simulations which attempts to model human behaviour sufficiently so that the forces will take some actions automatically (without requiring man-in-the-loop interaction).”

(M & S Office, U.S. Army, 2011)

CGF are the automated forces provided in the simulations, whether ally or enemy. Sometimes the simulators themselves are also referred to as CGF. They are an attempt to model the behaviour of the human operators they replace. Techniques used to model and implement behaviour of these CGF are generally taken from other fields of work, e.g. Artificial Intelligence & Computer Games, where similar entities occur in different forms.

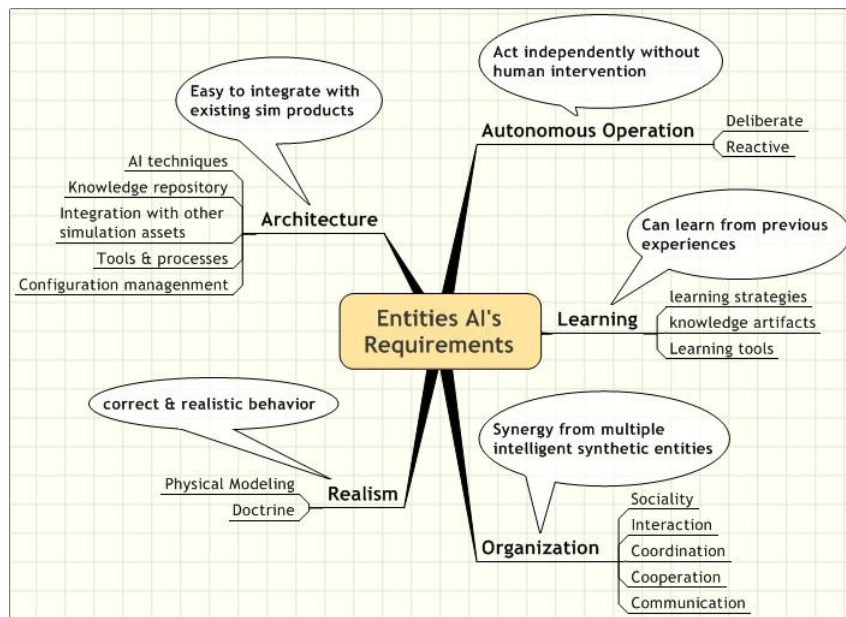
In the field of Artificial Intelligence, autonomic entities that model some behaviour are called (intelligent) agents. These can be used to automate a multitude of different entities such as elevators, washing machines and air traffic controllers (Rao and Georgeff, 1995). The agent is a model that should be able to reason on his knowledge and input from sensors, so that it can provide automatic output without the need for human interference. A lot of agent-architectures have already been researched and successfully implemented in industrial applications. Moreover, agents are used to model human behaviour realistically and effectively, so they are a good fit for substituting human operators in CGF. More information will be provided in Section 2.2.3. However, an agent is a much larger concept, applicable to almost all autonomic software (and even non-software phenomena). One could model an elevator as an agent, responding to stimuli from its environment with the action of moving up or down, opening doors et cetera. However,

an elevator is not generally taken as part of any military forces (whether computer generated or not).

In the field of Computer Games, the autonomic artificial entities are generally referred to as Non-Player Characters (NPC) or Bots. Real-time Strategy (RTS) games are similar to military simulations, since they both depict a battlefield filled with automated forces and opposing forces will fight a simulated battle if they encounter each other. As such, a game engine can be a decent starting ground for a simulator. However, the goals of simulations and games are very different. In computer games, excitement and fun have top-priority, generally opposing the realism that is needed for simulations. In contrast, the agent architectures from the AI community have realism as a high priority, with the ultimate challenge of mimicking (or surpassing) human intelligence. Furthermore, a connection between military simulations and intelligent agent architectures has been there from the start. Rao founded the prominent agent paradigm Belief-Desire-Intention (BDI) (Rao and Georgeff, 1991, 1995), and applied it soon thereafter in SWARMM (Rao, Lucas, Morley, Selvestrel and Murray, 1992), one of the first behaviour models supporting (air-)combat simulation.

For a more in-depth comparison between these different types (agent, bot, CGF), consult Sandercock et al. (2004).

### 2.2.2.1 CGF Requirements



**Figure 2.5:** Requirements for CGF AI as shown in Abdellaoui et al. (2009)

Numerous scientific articles have been written in the CGF domain, including some stating requirements that should be adhered to if CGF are to become realistic simulated entities (Tidhar et al., 1999, Brandolini et al., 2004, Abdellaoui et al., 2009). A recent set of requirements, from

Abdellaoui et al. (2009), can be found in Figure 2.5. They reason that requirements for entity AI in military simulation are not only realistic individual behaviour, but also the organization of multiple entities, the architecture possibilities, autonomous operation and learning. However, realism is still an important aspect and they divide it up in physical modeling and doctrine. In our project, physical modeling is left up to the simulator, but doctrine could be a new addition to the agents.

Brandolini et al. (2004) provide multiple additional motivational requirements such as a stress level indicator, survival instinct and moral/ethical motivations, providing extra realistic behaviour since doctrine training is not the only factor in human behaviour. Furthermore, the CGF's organizational requirements should include relevant military hierarchy, force aggregation and military report and feedback capabilities, similar to some of the requirements stated in Abdellaoui et al. (2009). Tidhar et al. (1999) also named similar concepts but in broader terms. This includes requirements such as emotions (motivational), social awareness (military hierarchy), explanations (feedback) and innovation (learning). Situational awareness or prediction capability is also named by Tidhar et al. (1999) and Abdellaoui et al. (2009), with the latter concluding that no such behaviour can yet be found in CGF.

#### 2.2.2.2 VR-Forces

VR-Forces is a simulation environment created by VT MÄK (2011*b*) specifically created to support CGF simulation. New versions of VR-Forces also provide a 3D view, but in the previous versions it only provided a top-down 2D view, similar to ISIS. In previous work on this TNO project, VR-Forces has been used as the CGF simulator and, while possible, this has not changed for this project. Moreover, Abdellaoui et al. (2009) provide a comparison of the currently available CGF and VR-Forces was found to be the overall winner of their evaluation. As such, it had scored best among its peers on the requirements from Figure 2.5.

**B-HAVE** The AI navigation module for VR-Forces is called B-HAVE (VT MÄK, 2011*a*). This means that B-HAVE takes care of the path-planning needs of the CGF. It is created using Autodesk's Kynapse technology<sup>4</sup> and plans paths for individual units.

**Unit Types** Units in VR-Forces will be set up based on the information provided in the MSDL ORBAT file and information provided in the scenario editor (the latter is still needed because enemy units are not setup with MSDL instructions). Since the agents will reason on the information sent from VR-Forces about these units, we will need to know the distinctions. First of all, a group in VR-Forces is called an *aggregate*, while an individual unit is an *entity*. So the military command echelons will be represented by an aggregate in VR-Forces and the equipment by entities. Aggregates have extra capabilities such as forming a formation, but they depend on their subordinate entities or aggregates to achieve their goals. Entities are capable of moving and firing their weapons on their own. Furthermore, VR-Forces is capable of simulating

---

<sup>4</sup>For more information, see <http://gameware.autodesk.com/kynapse>

many different types of these entities, like the tank M1A2<sup>5</sup>, or human soldier with a RPG-7<sup>6</sup> as weapon.

Updates concerning these aggregates and entities are sent in reports over HLA, thereafter intercepted by our agents.

**VR-Forces 4.03** During this project, an update has been applied to VR-Forces. Most of the implementation has been done using VR-Forces 3.12, the same Bronkers (2011) used, but with some conversion (on the VR-Forces plugin side of the project) the update has been accomplished as well.

### 2.2.3 Agents

Agent-based modeling is a modelling paradigm from the Artificial Intelligence field, used for autonomic entities that are capable of rational reasoning and communicating. As such, the weak notion of agency states that an agent's behaviour should be autonomous, responsive (reactive), pro-active and social (able to communicate) (Wooldridge and Jennings, 1995). However, as shall be seen later on with the Command Agents, not all agents adhere even to this weak notion. On the other hand many researchers have proposed more specific requirements in particular domains. In the domain of military simulation, Lucas and Goss (1999) provide a list of key characteristics for intelligent agents, as shown below:

- autonomy
- high-level representation of behaviour - easy to define command and control architectures
- flexible behaviour, combination of pro-activity and reactivity
- real-time performance
- suitability for distributed applications
- ability to work cooperatively in teams

This list already entails part of the criteria required for the CGF, thus agent-based technology should be an excellent fit. Next to translating the orders to an understandable level for the simulator, agents can be used to obtain more of the AI requirements from Section 2.2.2.1.

There are several different agent paradigms, the one used by TNO to support the CGF is the BDI agent. The Norwegian research institute FFI implements a Context-Based Reasoning (CxBR) agent system instead, which should allow for future comparison of agent paradigms. This also shows that similar behaviour might be implemented with different paradigms, with the only difference being the way in which it is written down. Section 2.2.3.1 provides an explanation of the BDI paradigm.

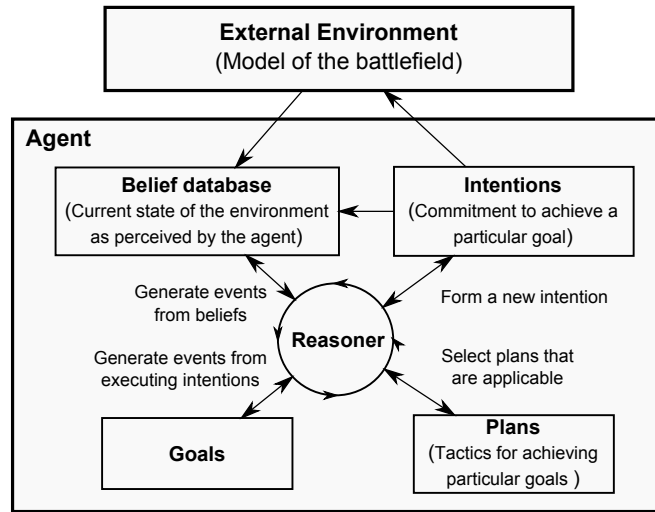


Figure 2.6: *BDI reasoning framework.*

### 2.2.3.1 Belief-Desire-Intention (BDI) model

Here we will discuss the Belief-Desire-Intention model (BDI) from Rao and Georgeff (1995), implemented by TNO in their Command Agents. This model has been used extensively in industrial and scientific applications and frameworks implementing BDI are numerous. BDI is a paradigm for (agent) software engineering with its roots in folk-psychology. Using beliefs about the environment and desires about what should be achieved, BDI provides the means for agents to be modelled in a way related to human thinking. See Figure 2.6 for an abstract representation of reasoning in a BDI agent (represented by the box). Here observations from the external world (the simulator) are turned into beliefs, after which internally goals (desires) are triggered and chosen. The agent then has to possess some (pre-built) plans that can fulfill the chosen goal. One of these plans (tactics) will be picked by the reasoning engine and this plan will initiate an intention, either changing a belief or executing an action in the external environment.

For example, assume an obedient BDI agent has a desire to execute every order it receives. When a commander then provides an order, the agent will believe it has received an order. Because it is an obedient BDI agent, it will want to execute the order it received, thereby creating the intention to execute the order. If the agent then believes a plan exists that can currently accomplish this intention, (the first action in) this plan will be executed.

Further in this section I shall explain the different elements of BDI in more detail, provide a review of this paradigm and discuss a framework, used for the Command Agents, that implements this paradigm.

<sup>5</sup><http://en.wikipedia.org/wiki/M1a2>

<sup>6</sup><http://en.wikipedia.org/wiki/RPG-7>

**Beliefs** Knowledge gained by the agent from reasoning or sensing are stored as *beliefs*: these are the facts the agent subjectively believes to be true. An agent can respond to new beliefs, e.g. by performing actions. Thus beliefs allow for the agent to react to its environment, an important feature.

**Desires** However, the agent can also act in a goal-driven fashion because of the agent's *desires*, which are the goals that the agent wants to achieve. These desires might be defined in the form of multiple sub-desires / sub-goals to be achieved or basic actions to be performed. Desires can also be triggered or deactivated based on certain beliefs and they are generally accomplished by executing an action or adjusting a belief.

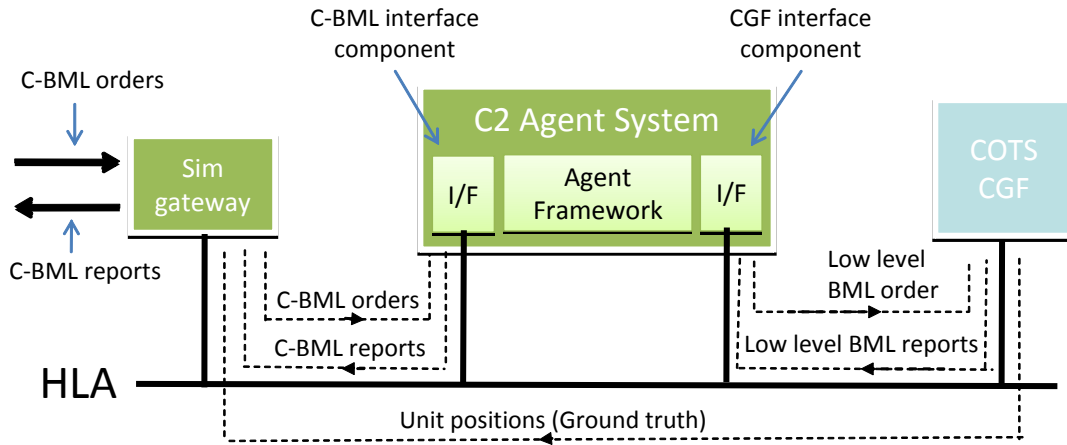
**Intentions** The third element of the BDI framework is the *intention*, the deliberative state of the agent. When the agent has a desire and believes it knows a plan (a sequence of actions to achieve this desire) it will create the intention of performing this plan. Then, when the agent believes that the current environment allows the intention to be fulfilled, it will perform the first intended action. This extra step allows for the agent system to commit to its course of actions as late as possible, also called the least-commitment approach. This allows the agent to respond to changes in its environment occurring while performing an action, something that is not possible in most planning methods. Thus in an ever-changing environment with imperfect information and bounded resources, the BDI agent can still function properly.

**Review** BDI has been used in military applications numerous times already, e.g. (van Doesburg et al., 2005, Mcilroy et al., 1996, Tidhar et al., 1999, Rao et al., 1992), and there are many extensions available as well. Having ample examples and frameworks to build on makes it easier to develop a military application with BDI agents. Another positive point is that BDI agents can reason in real-time because the plans used are provided in advance, yet they are still adaptable because plans are only chosen on the last possible moment. The representation using beliefs, desires and intentions provides a natural way of modelling human decision making, making it easier to get the required CGF behaviour. Easy creation of CGF behaviour by using plans allows for the easy addition of doctrine, one of the requirements from Abdellaoui et al. (2009). This should make the BDI agent an excellent addition to the simulation environment.

**Jadex** Jadex is a BDI framework in which agents can be modelled using Java and XML files. Instead of intentions (and actions), Jadex provides the agents with plans, in which the sequence of actions to be taken / modifications to be made (internal or external) are written. The XML files are a description of an agent's beliefs, goals (Jadex's desires) and plans and how they interconnect. Multiple (similar) agents can be created based on one XML file, or multiple agents can each have their own XML file. This allows, for example, an easy creation of multiple Platoon agents, which have their own specific set of beliefs, but share the same reasoning capabilities as the other Platoon agents.

## 2.2.4 TNO Architectures

### 2.2.4.1 System Architecture



**Figure 2.7:** System architecture (Bronkers et al., 2011)

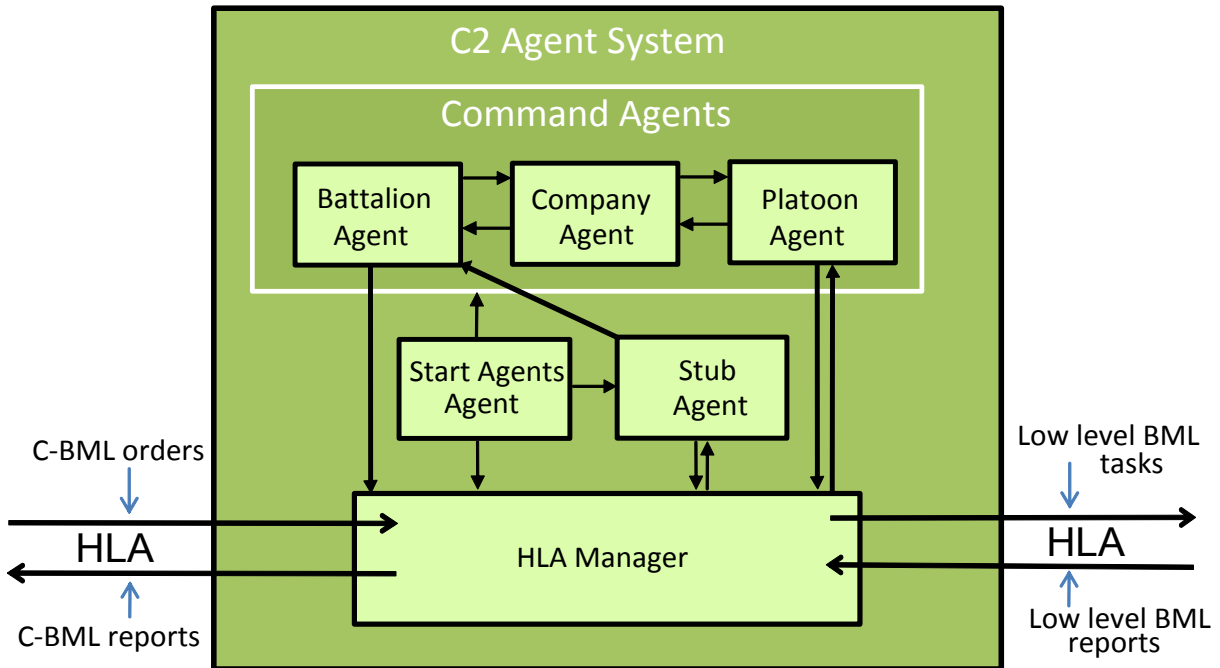
The system architecture created during the previous TNO project (Bronkers, 2011) is shown in Figure 2.7. Note that ISIS would be located at the left end of Figure 2.7, sending C-BML orders and receiving C-BML reports from the Sim gateway. The *Sim gateway* is the communication link between the current C2IS (ISIS) and the simulation (agent system and CGF). C-BML orders come in from ISIS in the Sim gateway, after which they are sent to the C2 Agent System, which is discussed in more detail in Section 2.2.4.2. The C2 Agent System outputs the Low level BML orders to the current CGF (VR-Forces), so that the simulation can take place. During the simulation, reports, e.g. the successful completion of an order or the damage received by a unit, are sent to the agents and unit positions are sent to ISIS from the CGF. The C2 Agent System then translates these low-level orders to high-level (battalion) reports which are finally provided to the gateway.

### 2.2.4.2 Agent Architecture

The C2 multi-agent system created by Bronkers (2011) consists of 6 BDI agents, which are shown in Figure 2.8. The middle 2 agents are purely functional: the StartAgents agent and the Stub agent.

**StartAgents agent** The StartAgents agent is a functional agent only used in Bronkers (2011) to initiate all the other agents needed in the current scenario. In this way, only one agent has to be started in Jadex. In the same way, all agents can be stopped by sending a message to the StartAgents agent.





**Figure 2.8:** Agent architecture as used in Bronkers et al. (2011), Bronkers (2011).

**Stub agent** The Stub agent is a functional agent created by Bronkers (2011) for increased testing speed. It subscribes to the BML orders received by the HLA Manager. However, the Stub agent also allows the developer to send predefined orders to the C2 Agent System without needing a connection with ISIS. The idea is that the Stub agent can also be used for further interpreting the BML orders before they are sent to the battalion agent. This might, for example, be useful when the BML order consists of multiple orders and these have to be sent individually to the Battalion agent. The Stub agent also receives reports from the Battalion agent and sends these to the HLA Manager.

**HLA Manager agent** The HLA Manager agent is the communication agent on both ends of the C2 Agent System, sending messages to and receiving messages from HLA. Different agents subscribe to certain information from the HLA manager, e.g. a Platoon wants to receive low-level BML reports, and the HLA manager will forward messages of this type to subscribed agents. The HLA Manager will also send acquired reports over HLA to ISIS. For example, when an entity changes its position in VR-Forces, the entity will be forwarded to HLA, from where the HLA Manager intercepts the message. In the HLA Manager, this entity will be cast into one of the applicable Java classes, e.g. the `GroundVehicleObject`, and forwarded to the Platoon agent, who will store or update this entity in its beliefs.

**Tank Battalion agent** The Tank Battalion agent represents the commander of a battalion and listens to the orders provided by the Stub agent. Based on the name provided by the StartAgents

agent and the ORBAT from the MSDL file, the Tank Battalion agent also gets a number of Company agents assigned to it. After receiving an order, the Tank Battalion agent will delegate it to its subordinate Company agents whenever appropriate. However, additional reasoning might also be used for more complex orders. Once the Tank Battalion agent receives the same task report from every of its subordinate Company agents, it will send this report to the Stub agent.

**Tank Company agent** The Tank Company agent represents the commander of a company unit and receives commands from a Tank Battalion agent. These commands are then delegated to its subordinate Platoon agents, provided to the Tank Company agent by the StartAgents agent upon initialization. Similar to the Tank Battalion agent, additional reasoning is used to form the orders it sends towards the Platoon agents. Moreover, the Tank Company agent will send a task report to its superior once it has received one from all of its Tank Platoon agents.

**Tank Platoon agent** The Tank Platoon agents used by Bronkers (2011) are on a direct level of communication with the *aggregate* unit in VR-Forces. Previously, the Platoon agent had to send orders for individual tanks to VR-Forces, but an update to the simulator allowed commands on the aggregate level (a combination of tanks, the platoon level for VR-Forces). This means that the Platoon agent will provide the simulator with the actual low-level commands that it needs to execute. The Tank Platoon agent will subscribe to the reports sent from VR-Forces to the HLA Manager and report these to its superior Company agent.

## 2.3 Decision Support System for Command & Control

”A DSS is an interactive computer-based system intended to help decision-makers utilize data and models to identify problems and make decisions.”  
(Sprague Jr, 1980)

Some work related to our DSS for C2 is shown in this section, indicating the possibilities of such a system and possible requirements or lessons that we can reuse.

### 2.3.1 Previous C2 DSS projects in literature

There have been a multitude of decision support systems for the C2 domain in countries and military forces all around the world, some of which may be in use, discarded or still in development. We shall discuss a few recent systems and what their plans or successes are, showcasing possible use-cases for a DSS in C2.

First and with the grandest imagination, Deep Green (Surdu and Kittka, 2008*a,b*) is a project from the Defense Advanced Research Projects Agency (DARPA). As seen in Figure 2.9, Deep Green is to be a decision support system that helps the commander by analyzing possible future states of the battlefield,



Figure 2.9: DARPA's Deep Green concept

using "Blitzkrieg", dependent on the COAs of friendly and enemy forces. Only those future states that are becoming more likely based on the current state, updated using "Crystal Ball", should be developed further. This should allow for *anticipatory planning*, so that a new COA is ready before the situation occurs. As stated, this is as of yet only envisioned, but it does indicate the future vision of C2-Simulation-DSS interoperability. Deep Green is supposed to support the commander during mission execution with the use of simulation, without intervention needed from a staff of experts. The decision support system will intelligently plan simulation sessions and retrieve its results based on the current state so that the commander can access possible future states and be notified if new game-changing opportunities arrive for either himself or his enemy. The main difference between DARPA's Deep Green and past projects is how this DSS is to support during mission execution instead of merely be used as training help or mission preparation. Hereby it showcases the possibilities and future use of DSSs as they are created in projects like ours.

Other recent projects have provided results of their experiments, possible useful information. One of these is Herbinet et al. (2010), who describe a recent experiment in C2 & Simulation coupling performed by the Commission Electronique et Optronique Ad-Hoc Working Group Modeling and Simulation (COMELEC AHWG M&S), a cooperation between the French and German military forces. Being a part of the MSG-085 group, they were also tasked with validating the use of C-BML in C2-simulation interoperability. According to Herbinet et al. (2010), one of the use-cases in the experiment was *Planning*. In this use-case, French and German officers created 3 friendly COAs that were all simulated by their CGF, APLET, against one predefined enemy COA. Apparently, the second COA was chosen as the best one. More importantly, Herbinet et al. (2010) conclude that the officers reported both a lack of control and a lack of information. This has been attributed to a lack of explanation from the agents; a limited number of possible orders and information used by the agents; being designed for a lower command hierarchy level; non-standardized graphical symbols across the different used systems; employment of a C2 surrogate system; and a GUI not designed for operational use. Most of these problems inevitably seem to be related to the system being a prototype, as ours shall be, such as the limited number of orders and user interface. However, other problems such as the graphical symbols, C2 system and command hierarchy seem solvable in our project. This project showcases the use of a DSS (the simulator) in C2 as a help during the analysis and comparison of COAs, as preparation for a mission.

CASA (Hanna et al., 2005), or Course of Action Simulation and Analysis, shows an effort that has been made for a DSS in C2 for comparison of COAs, based on Effect-based Operations (EBO). EBO is the discipline focussed on an abstraction of effects where not only direct military effects cause a mission to be accomplished. Instead, secondary and tertiary effects are to be considered where the lower levels cause cascading effects in the later levels. For example, the commander's intent might be to get the support of the local population in an area. One course of action to fulfill the commander's intent might be to disrupt the enemy's influence over the town by destroying the enemy's stronghold. However, repairing a destroyed bridge near a village might cause a major improvement in the local economy and that might cause a shift in the town's support of enemy forces towards support of our own forces. CASA has created an ontology where the commander's intent can be broken down into smaller measures, until at the bottom low level

characteristics (or actions) can be found that are directly measurable in the environment and that can be aggregated (and weighted) to describe any measure needed at higher levels. This should allow means for comparison of disparate COAs for the same commander's intent, as in our example. The focus of the project is on the effect the commander's intent has on the value of actions that are executed. Destroying all enemy resistance is not as important in all plans. Considering secondary and tertiary effects focusses greatly on the motivational part of entities as described previously in CGF requirements (Section 2.2.2.1). Show of heavy force might cause opponents to be scared and flee instead of fight. In the end, they conclude that the ontology is too limited for a fully developed system, partly because displaying the large amounts of data affecting the COA results created a cluttered and hard to navigate DSS GUI. Thus, for supporting COA analysis and comparison the DSS should provide a well-founded score, but it is also vital that it presents this information in a usable way. This project shows us another side to possibilities of COA analysis and the pitfalls it brings along, as once again the problem seems to be a DSS GUI that does not match well with the needs of the C2 process.

Back in 2000, Shu-hsien and Liao developed a learning framework for a DSS for C2. The commander's intent, based on the current situation, was classified as one of the predefined cases. At that point the user would work through the MDMP and create COAs for wargaming. If there was a significant difference in implementation at the end of the decision-making process, a new case could be created. No further information is given about specifications of the DSS. The system was tested on 200 Taiwanese officers, where one group would use the DSS and the other would not, and the results showed that those that used the system had an advantage in the wargaming scenario compared to those who did not. This DSS showcases another use-case for a DSS in C2, as it does not help in analysing COAs like the previous ones, but provides an interpretation of the commander's intent. One of the main differences is of course that no simulation is used in this DSS, yet the DSS still provides additional value as the experiment showed.

As is shown by this subset of DSSs for the command & control domain, there is a great application area in the MDMP of C2 for such systems. Be it as preparation before creation of a COA, as help during analysis and comparison of COAs or as support during the execution and real-time adaption of a COA, DSSs show promise. However, the use of simulation to support the DSS is assuredly in the section of COA analysis, on which a DSS can be built for COA comparison. As such, we shall follow in the footsteps of Hanna et al. (2005) and Herbinet et al. (2010) to provide TNO a prototype DSS for COA comparison.

### 2.3.2 DSS Requirements

"Tools must complement human COA development and analysis, not try to replace it."  
(Hazen, 2011)

Another subset of the literature on command & control systems discusses the use and the requirements of previously discussed DSS systems for C2, such as Davidson and Pogel (2010), Lafond et al. (2010), Hazen (2011) and Prelicean et al. (2010). Since we are trying to create our own prototype DSS, we should learn from these requirements.

Most importantly, according to Lafond et al. (2010), DSS might do more harm than good if they are not in tune with the cognitive aspects in C2. This is similar to what Hazen (2011) describes in the quote above, who writes about the challenges that are faced with the C2-simulation interoperability. According to Hazen, there is a mismatch between what simulations have provided and what C2 needs. Generally, the simulations do not provide enough added value to the already-expert knowledge of the commanders, or the simulations take too much time to setup and/or run. Hazen also provides a table in which combinations are stated of C2 requirements and the simulation research needed to accomplish these. One of the main points concerns standardization, using research like CBML and MSDL, which are being tested in the our current project. Other points concern the interface and usability of the tool, the speed, the transfer of experiential knowledge, the addition of non-military scenario information, extra visualization tools, tools that work across all platforms and a better explanation of the effects that input parameters have on the simulation. Mainly, Hazen states that "tools must complement human COA development and analysis, not try to replace it", providing possibilities for the use of simulation if they:

- Fit the timescale of decision-making;
- Incorporate and provide access to experiential knowledge as it is developed by other commanders;
- Provide value added such as exploratory analysis of variations on candidate COA;
- Provide results that are intuitively understandable by military operators, and are credible in known situations;
- Provide means of identifying important decision points and conditions to assist in the development of Commander Information Requirements.

Another useful paper is Prelipcean et al. (2010), who provide us with pointers towards designing a decision support tool for course of action analysis, exactly what we are aiming at. They reiterate that the DSS should support the following functions:

- Description of the event;
- Development and description of possible COAs;
- Identification of criteria to be used in the evaluation process;
- Evaluation of the COAs according to the selected criteria;
- Analysis and comparison of these COAs;
- Post-execution analysis.

Description of the event includes assumptions about the enemy forces and COAs and own forces capabilities et cetera, which can all be modelled in the MSDL standard that we are going to use. Development and description of the candidate COAs is already possible in our setup thanks to

our C2 connection and the C-BML standard for communicating this COA. The other steps are of value to our DSS model, as we will need criteria for the comparison, we will need to simulate the COAs and evaluate them according to the chosen criteria. Furthermore, the DSS will need to allow the comparison of these results.

Criteria related to these supporting functions are described by Prelipcean et al. (2010) as such:

- Show all the assumptions taken;
- Integrate with other information, planning and decision systems;
- Enable creation of new COA;
- Enable updating existing COA;
- Enable verification of COA feasibility;
- Propose different criteria to be considered in the evaluation process;
- Allow staff to choose other criteria interactively;
- Show all information about the quality of each COA together with a ranking;
- Allow dominance check ;
- Allow post-analysis facility for feedback from officer about how effective the CoA really was (learning);
- Allow user-profiles for different staff;
- Provide natural interface;
- Users should be aware of the limitations and the level of trust;
- Explanation facility providing explanations adapted to user's knowledge/background/experience and time available;

Certainly not all of these criteria can be met in this project, but they do guide us in a certain direction. Interesting for the usability of our DSS is the interaction requirements that the staff should be able to select criteria in a natural interface, which coincides with the requirement from Hazen (2011) that the humans should not be replaced but supported.

## Chapter 3

# Methods

In this chapter, we shall describe how we set up our research, what tools we used and how we are going to answer the research questions.

### 3.1 Expert Information Solicitation

First we shall discuss how we got the expert information needed to properly implement military tactical doctrine into the agent and what our DSS should be like. As can be seen in Section 2.3, literature provides ample discussion on the needs of a DSS in the C2 domain. However, TNO is specifically interested in what the RNLA, the stakeholder of this research, would like to see. For this reason, we have held meetings with subject matter experts (SMEs), related to the RNLA, on different occasions to discuss the possibilities for this project and what options would be preferred. Below, a quick overview will be given for every meeting, shortly explaining what was discussed and concluded.

#### Brainstorm session - November 9th, 2011

On Wednesday November 9th from 10:30 till 13:00, we held a brainstorm session with 4 SMEs from RNLA's Simulation Center (SimCen). The purpose of this session was to find out what kind of DSS the RNLA would be interested in. Our idea of a DSS in table-form was confirmed, in which the rows would indicate different routes to be compared and the columns would provide different measures of effectiveness (MOEs). Furthermore, a number of MOEs were provided by the SMEs.

#### Discussion - November 22th, 2011

In a discussion with 1 SME from TNO, on Tuesday November 22th from 13:00 till 15:00, we asked for feedback on the Artificial Intelligence possibilities in the DSS and how interesting these might be to potential customers. We proposed the idea of either intelligent path planning or intelligent agent behaviour. It was concluded that intelligent agent behaviour might be more promising to start with.

#### Discussion - December 9th, 2011

In a later discussion with the same SME, on Friday December 9th from 10:30 till 12:00, we talked about what intelligent agent behaviour should be implemented. Having access to several Combat Instruction Sets (CIS), we proposed three options: Assault, Blocking Position or Formations. In the end, we concluded that we should start with Assault, working backwards to a Road March (using Formations) that could then be interrupted, allowing the Blocking Position to be used. This should allow modular development of agent behaviour.

#### Discussion - May 15th, 2012

At this point, we had implemented the basic behaviour shell from the provided CISs as far as we could determine. However, the CIS descriptions do not discuss every detail, so assumptions



had to be made during the implementation. Therefore, we traveled to RNLA's SimCen in Amersfoort again, to find answers to a set of pending questions on this matter. We had a discussion with 2 SME's from the earlier session on November 9th, and later in the day another discussion with a new SME who had played a part in the creation of the CISs. Based on these discussions, we implemented the final parts of the agent behaviour.

## 3.2 Hardware & Software

All the software related to the simulation was installed on a Dell XPS M1710 laptop with an Intel Centrino Duo dualcore-processor, running Windows XP Professional. However, when we used VR-Forces 4.0.3 instead of 3.12, we needed to use 2 of these laptops, where one ran the simulator and the other ran the agents and RTI to connect both.

**Jadex** Because we are extending the BDI agents created by Bronkers in Jadex<sup>1</sup>, we have also used the Jadex framework to implement these extensions. For more information on Jadex, consult Section 2.2.3.1.

**Java** The DSS was created using the Java programming language. Furthermore, the agent's plans in the Jadex environment are also written in Java.

**Eclipse** The source code of our agents and of the DSS was edited using the open source Integrated Development Environment (IDE) Eclipse<sup>2</sup>.

**C++** The plugin of the simulator requires code in C++.

**Microsoft Visual Studio 2008** The simulator's plugin has been built under Microsoft Visual Studio 2008.

## 3.3 Validation

To answer our research questions, we will validate our prototype's behaviour. There is no time in this project to perform experiments on the cognitive usability aspects of the decision support system or for expert's review of the DSS's usability and simulation's realism. Instead, we will compare our resulting demo of both the simulator and the decision support system to the requirements discussed in Chapter 2. Further validation will be required as one of the next steps in future work.

---

<sup>1</sup><http://jadex-agents.informatik.uni-hamburg.de/>

<sup>2</sup><http://www.eclipse.org/>

## Chapter 4

# Model

In this chapter, we will discuss the model we have designed for this Master's project and how it has been implemented. We will chronologically guide you through the development of this model: first we show the extension we chose for our agents in Section 4.1, then in Section 4.2 we show how our new system architecture works out, followed by the new agent model in Section 4.3 and finally the model for the decision support system in Section 4.4.

## 4.1 Increasing Realism

As discussed in Section 2.3, we have chosen to focus our decision support system on analysing simulated courses of action (COAs). To this end, the Command Agents connecting the C2 system and simulator are to be used to compare measures of effectiveness (MOE), or criteria, of multiple routes, providing feedback to the commander about the simulated results. This should provide the commander with a quick overview of the possible benefits of each route. However, before we can provide a realistic overview, we need a realistic simulation.

We have identified three possibilities in this setting for increasing the realism:

- Agent Behaviour
- Path Planning
- Enemy Behaviour

There is a certain overlap between these three and in the end all will be needed for a realistic simulation. However, due to time constraints, the current project can only focus on (a subsection of) one. After a discussion with a TNO SME (see Section 3.1), we decided to focus on Agent Behaviour. The other ideas are thus for future work, however, initial ideas can be found in Appendix A.

### 4.1.1 Agent Behaviour

Currently, the agents only act as order distributors, providing a simplified order from the commander to the units in VR-Forces. Instead, we would like to refine the behaviour of the agents (and the units in VR-Forces), to enable more autonomy. The subordinate commanders (agents) should provide intelligent implementations of the commander's order. To this end, extra knowledge of doctrine and formations can be added, allowing the units to react more realistically in the simulation. For example, when currently fired upon, a company will keep driving towards its goal while shooting back at the enemy. Instead, we might like to see the company backtrack to a previously safe position to devise a new plan, based on a.o. the enemy's expected power, current mission objectives and terrain properties. Furthermore, the new plan should depend on the commander's intent: is the goal to get to a location as fast as possible or to clear the whole area of enemy resistance. This change in behaviour would effect the MOEs of a chosen route in a more realistic manner, e.g. increasing the amount of time but decreasing the amount of casualties taken when enemy is encountered.

By making the units respond to the environment (more realistically), the possibility of wargaming within the simulation might also be enabled. For example, the user could add

enemy artillery during the simulation, to see how that would impact the MOEs of the current route. Currently the agents would not respond to this, while we might want them to devise a new plan to stay out of reach of the artillery. As such, we will implement a set of doctrines that showcase a variety of realistic agent behaviour like planning, reacting and coordinating.

#### 4.1.2 Combat Instruction Sets (CISs)

Documents designed for TACTIS<sup>1</sup> are available that describe military doctrine on the company and platoon level, which can be turned into intelligent agent behaviour. Such a document is called a Combat Instruction Set (CIS). The agent will then reason on when to use what CIS based on its beliefs about the environment and its own mission. In a second discussion with the TNO SME, see Section 3.1, we decided upon a specific subset of these CISs, which will be explained next. For a list of all the CISs used during this project, consult Appendix B. We shall describe the main CISs in more detail further on in this section.

We have chosen 3 main CISs to be implemented:

1. Assault an enemy position (short: "Assault")
2. Tactical Road March (short: "March")
3. Hasty occupation of a blocking position (short: "Blocking Position")

The Assault showcases how a seize (the most intelligent behaviour the agents could provide before this project started) is realistically performed. The March allows our agents to move realistically along the terrain with coordination, unlike the individualistic movement it showcased before. The Blocking Position finally provides reactive behaviour that has not been seen before, as the military forces react to enemy behaviour.

##### 4.1.2.1 Scenario

These CISs were fit together in the following scenario, which will be implemented for our final demo:

A company is tasked with an Assault. This company will March towards a start line, where it will initiate the Assault. During the Assault, unexpected enemy behaviour is discovered and this company is tasked to create a Blocking Position against this enemy. Once the enemy is dealt with, the company will continue on its Assault and seize and destroy the enemy positions.

Further in this section we shall provide descriptions of these main CISs as we interpreted them. Each CIS description will contain the general description as provided in the CISs, a summary of the CIS as we interpreted it, figures concerning the CIS and possible sub-CISs used by the CIS.

---

<sup>1</sup>Tactical Indoor Simulation from Thales. Tailored for combined forces instruction and training, see [http://www.thalesgroup.com/Portfolio/Defence/D3S\\_product\\_simu\\_tactis/?pid=1568](http://www.thalesgroup.com/Portfolio/Defence/D3S_product_simu_tactis/?pid=1568)

#### 4.1.2.2 Assault

Our current assault is uncoordinated: single units (tanks) move towards their destination at full speed while firing at enemy targets. However, to minimize casualties taken, maximize damage done and allow more realism in the simulation, military doctrine should be followed. Among the CISs we found the CIS "Assault an Enemy Position" that describes doctrine during such an assault. Although this CIS is on the platoon level and there was no equivalent on the company level, we assume a linear transformation to the company level. This is supported by the description stating that platoons rarely assault individually.

This Section is divided into the following sub-sections: first, a general description of the task is provided as they are written in the CISs; this is followed by a summary of the information provided in the CIS, as it is applicable to our agents and we interpret it. Finally, all Sub-CISs are named that are used during the Assault.

##### General description of task

*The assault is the actual overrunning and seizing of an enemy position. The platoon is operating in an assault as part of a company team and will hardly operate independently because of its limited combat power. To be successful the combat ratio of an assault has to be 3 : 1, so a company team can assault a (reinforced) enemy platoon. Basically the assault will take place by fire and movement of the company team, supported by indirect fire and/or air support. After crossing or on the start line ( $\pm 1500$  m from objective) at H-hours, the platoons will disperse (line formation) and move forward (traveling) in the direction of the objective. When enemy is firing, the company team will move forward by fire and movement (platoon bounds and/or section bounds). When the final phase of the assault begins ( $\pm 600$  m from objective), all vehicles move with maximum speed in the enemy direction while firing on enemy positions. Tanks will cross the objective and take in positions 300 – 500 m beyond it, to secure the seized objective from counter attacks. Aifvs will dismount aggressively at the edge of the objective and vehicles and firing teams will move forward by fire and movement. After cleansing the objective the armored infantry platoon(s) will also take in positions 300 – 500 m beyond the objective and, with the tank platoon(s) consolidate and reorganize.*

##### Assault an enemy position - Mechanized Infantry Platoon

*An assault is the culmination of an attack that closes with and destroys the enemy; it is the actual overrunning and seizing of an occupied enemy position. Normally the assault force is composed of tanks and infantry elements under the control of company (team) commander. The assault force moves by fire and movement (See CIS "Platoon fire and movement") along covered and concealed routes to the enemy position until it reaches the assault position (the last covered and concealed position before the objective). Typically, tanks lead, with Aifvs following to protect against dismounted infantry and provide suppression to the flanks. The assault elements move rapidly in a line formation, under cover of direct and indirect fires to the*

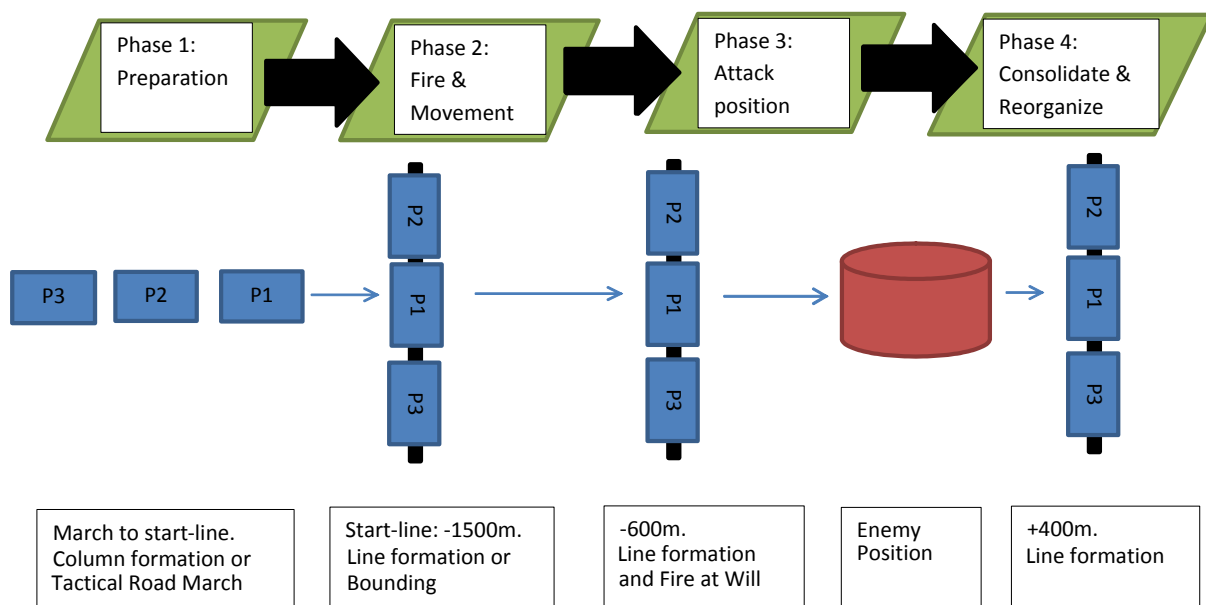
*objective, moving through it to the far side and consolidate and reorganize. Infantry dismounts to mop up resistance and clear objective.*

Assault an enemy position - Tank Platoon

## Summary

See Figure 4.1 for a schematic view of this CIS. Below is a summary of our interpretation of this CIS.

- Preconditions
  - Combat ratio of 3:1,
  - Start-line at  $\pm 1500$  m from objective.
- Phase 1: Preparation
  - Company moves towards the assault start-line (or to 1500 m from objective).
  - At the start-line, company and its platoons move into Line formations.
- Phase 2: Fire & Movement
  - Company executes CIS – Line formation traveling.
    - \* Company moves to 600 m from objective in Line formation.
  - When fired upon, company executes CIS – Fire and movement instead.
    - \* Company and/or its platoons move with bounds instead of in one line.
    - \* Successive bounds: one element bounds towards the next position and is followed by the rest after a time period.
    - \* Company/Platoon bounds: the whole company or platoon bounds towards the next position at the same time.
  - Bounding along covered and concealed routes.
- Phase 3: Attack objective
  - Company executes CIS – Execute final phase of attack.
    - \* Company moves in Line formation (for maximum fire power in front),
    - \* Company moves at maximum speed towards objective,
    - \* Company fires at will on enemy positions.
  - Infantry mops up remaining resistance at the objective area.
- Phase 4: Consolidate & Reorganize
  - Company executes CIS – Consolidate and reorganize.
    - \* Company crosses objective and takes in positions 300 – 500 m beyond it.
    - \* Platoons execute CIS – Hasty occupation of a battle position.



**Figure 4.1:** Schematic overview of the Assault an enemy position CIS, divided in 4 phases. The top of the figure shows the current phase, the middle shows a schematic view of the platoons during this phase, the bottom shows a description of this schematic view.

### Sub-CISs

This CIS consists of the execution of the following CISs:

- Assault an Enemy Position (Platoon CIS available<sup>2</sup>)
  - Fire and Movement (Platoon CIS available<sup>2</sup>)
    - \* Line Formation Traveling (Company & Platoon CIS available), or
    - \* Line Formation Traveling Platoon/Team Bounds (Company & Platoon CIS available), or
    - \* Line Formation Traveling Successive Bounds (Company & Platoon CIS available)
  - Final Phase of Attack (Platoon CIS available<sup>2</sup>)
  - Consolidate and Reorganize (Platoon CIS available<sup>2</sup>)
    - \* Hasty Occupation of a Battle Position (Platoon CIS available<sup>2</sup>)

#### 4.1.2.3 March and Formations

The military also provides doctrine on movement towards objectives. As part of the scenario, the "Tactical Road March" CIS could be implemented to provide a more realistic coordinated

<sup>2</sup>Should be used as the Company CIS as well

mode of transportation. Currently all tanks are individually tasked by VR-Forces' B-HAVE module to move towards some position, yet behaviour in military scenarios is rarely on an individual basis. With these CISs, we hope to provide some coordination between tanks and platoons.

The Tactical Road March CIS is available for both the company level and the platoon level. The same is true for the formations that can be traveled in: the Line formation and the Column formation. We will provide the general descriptions as taken from the CISs and after that our own interpretation in the summary. The company CISs of the formations provide no information on when to use what formation, or what their use is. Instead only the platoon level description is provided for the formations.

#### General description of task

*A tactical road march is normally used to move a company (team) along roads from rear areas to assembly areas in preparation for a mission. A company (team) may conduct a tactical road march as a separate unit or as part of a battalion task force. A road march differs from other forms of movement in that the purpose is to relocate rapidly, not to gain or make contact, and it is conducted at fixed march speed and time intervals. The tactical road march is conducted when speed is essential, company (team) integrity must be maintained, road nets are available, and chance of enemy contact is limited.*

Tactical Road March - Company

*A tactical road march is normally used to move platoons along roads from rear areas to assembly areas in preparation for a mission. It differs from other forms of movement in that the purpose is to relocate rapidly, not to gain or make contact, and it is conducted at fixed march speed and time intervals. Although a platoon may be required to conduct an independent tactical road march, it normally moves as part of the company team. The tactical road march is conducted when speed is essential, platoon integrity must be maintained, road nets are available, and chance of enemy contact is limited.*

Tactical Road March - Platoon

*Formations are used to establish vehicle positions and sectors of responsibility during operations. The line formation traveling is used when the platoon:*

- *Is moving and is required to assault an enemy position/force.*
- *Has to engage an enemy force while the platoon is moving forward or backward.*
- *Has to cross open terrain or a danger area as quickly as possible with maximum firepower to the front.*

*The line formation traveling is formed with all vehicles continuously moving in a line, vehicles 2 and 3 are moving abreast and slightly behind vehicles 1 and 4.*

Line formation

*Formations are used to establish vehicle positions and sectors of responsibility during operations. The column formation traveling is used when enemy contact is not likely*



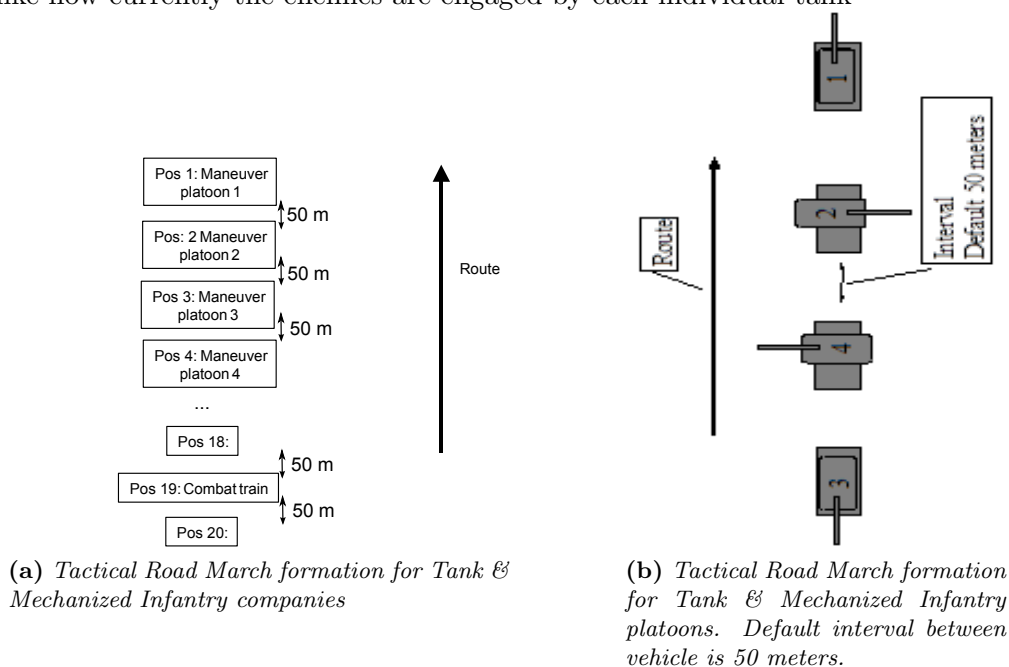
*and speed is essential. It is formed with all vehicles in the platoon in trail moving continuously. The formation provides excellent control and fire to the flanks but permits less fire to the front.* Column formation

Summary

**Tactical Road March** The Tactical Road March is the method of transportation used when no danger is expected and the company’s or platoon’s goal is to get to a location as soon as possible. In this formation, as can be seen in Figures 4.2a and 4.2b, all vehicles follow with a short 50 meters distance behind one another. This march is to be used to move towards the staging area in our demo scenario.

**Column formation** The Column formation is similar to the Tactical Road March, in fact the formations for platoons are the same as can be seen in Figure 4.3b. However, in the Column formation enemy resistance is not entirely unexpected while traveling. Therefore the first platoon in the column rides as a reconnaissance a great distance in front of the rest, as shown in Figure 4.3a. As a Tactical Road March is to be driven on a road, we would use the Column formation in our scenario when we drive towards a staging area without roads.

**Line formation** The Line formation is the army’s fighting formation against enemies who are engaged head-on, see Figures 4.4a & 4.4b. The firepower of the whole company or platoon is pointed to the front, while a formation like the column would only be able to point one tank at a frontal enemy. As such, the Line formation will be used to engage enemies as a collective, unlike how currently the enemies are engaged by each individual tank



**Figure 4.2:** The tactical road march formations for the different hierarchies.

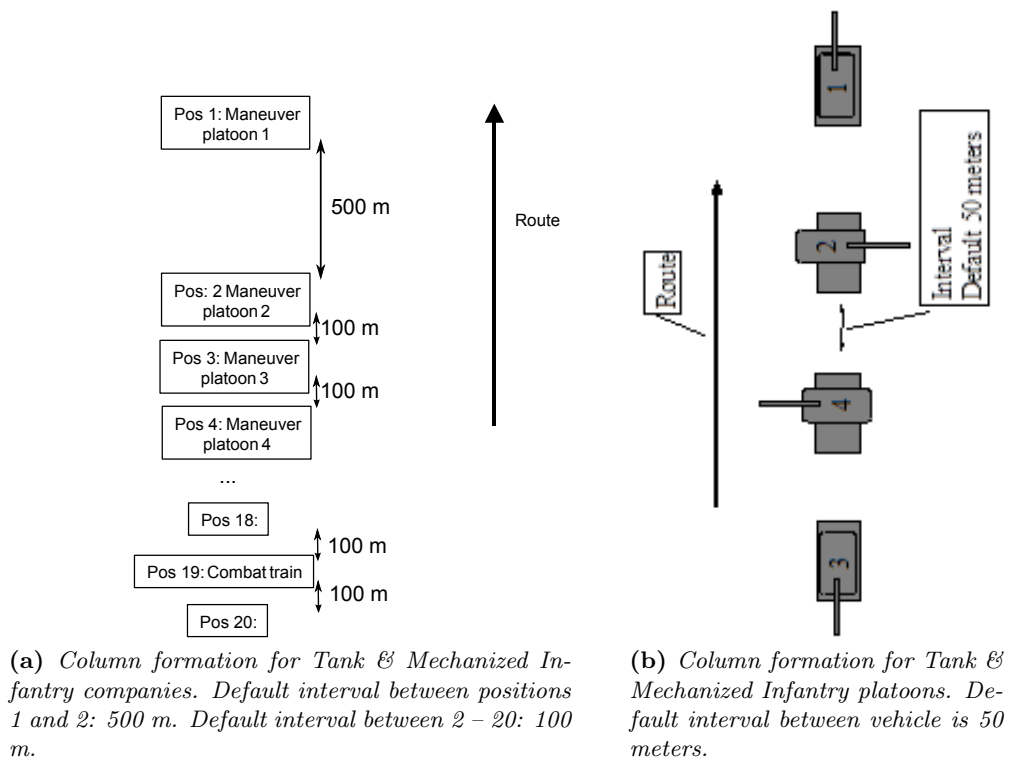


Figure 4.3: The column formations for the different hierarchies.

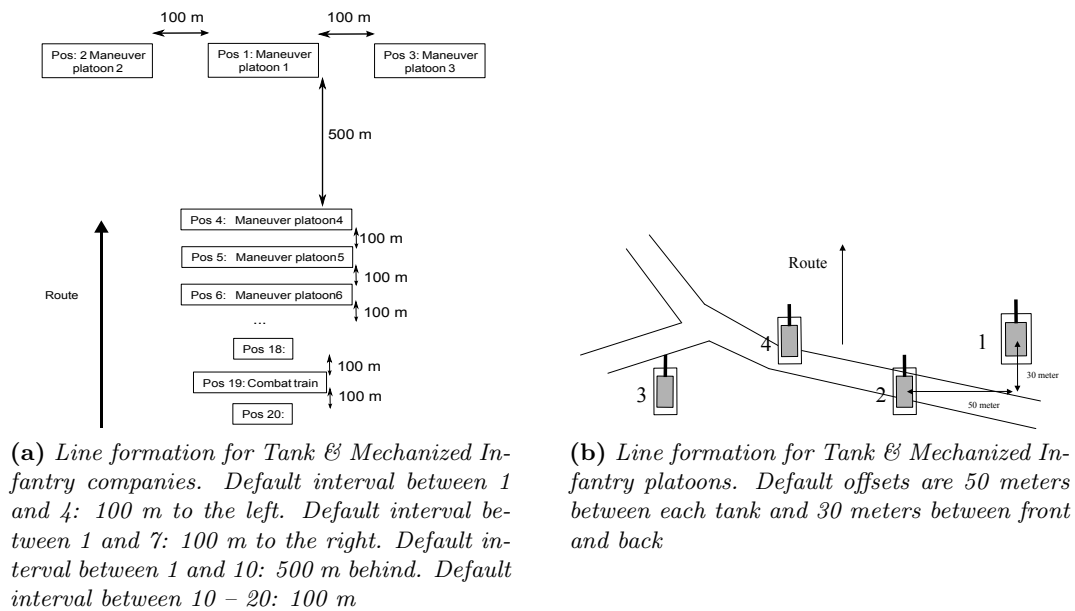


Figure 4.4: The line formations for the different hierarchies.

#### 4.1.2.4 Blocking Position

##### General description of task

*A blocking position is a shallow position in one direction. A blocking position obstructs the enemy's progress in a particular direction. In order to eliminate the enemy, the bulk of the company (team) means are positioned in front. Normally hasty occupation of a battle position is ordered to block not foreseen enemy actions.*

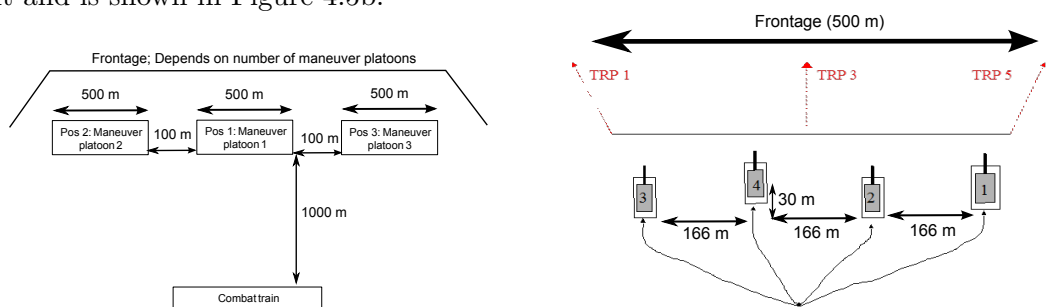
Hasty occupation of a blocking position - Company

*When the platoon is conducting offensive or defensive operations and it must defend when insufficient time exists to go through the stages of deliberate occupation, it establishes a hasty battle position. The vehicles of the platoon move from the platoon dispersion point to the firing position and occupy the hasty BP. Orient itself properly on the likely direction/avenue of enemy attack and/or assigned engagement area (EA) ensures survivability of the platoon and it's firing position.*

Hasty occupation of a battle position - Platoon

##### Summary

The "Hasty occupation of a blocking position" is another CIS that can provide visible intelligent agent behaviour. The hasty occupation of a blocking position is used to put a halt to the enemy's advance, when the enemy breaks through the defense line or if delaying combat is used. Either way, the blocking position is an attempt to save time by delaying the enemy. This could allow reinforcements to arrive in time. The delay is achieved because the enemy will need to unfold into its firing positions (e.g. line formation) to engage the blocking position. At this moment, the blocking unit should disengage and retreat to the following blocking position. In essence, the blocking position is the reverse "Fire and Movement" (phase from the Assault), eventually wearing down the enemy. In Figure 4.5 the positioning of the units during the blocking position is shown, which is in essence a stretched-out line formation. This CIS is on the company level, a platoon can not produce a blocking position on its own. Instead, a platoon is tasked with *Hasty occupation of a battle position*, which is also the task used for consolidation after the assault and is shown in Figure 4.5b.

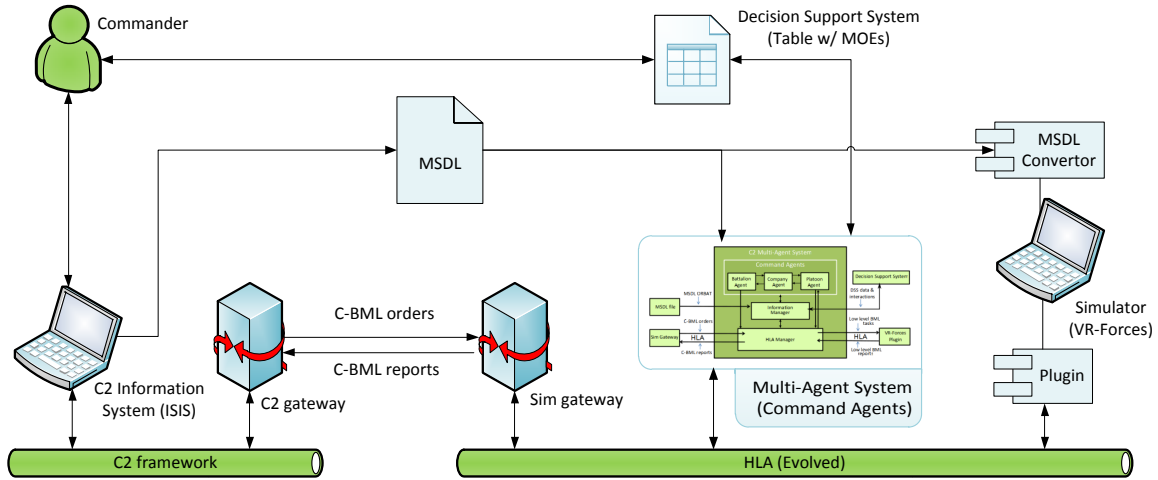


(a) Blocking position for company. Default frontage is 500 meters per platoon. Default distance between platoons is 100 meters.

(b) Battle position for platoon. Tanks are evenly divided over the frontage.

**Figure 4.5:** The blocking position formations for the different hierarchies.

## 4.2 System Architecture



**Figure 4.6:** *System architecture*

In Figure 4.6 you can see the new system architecture. Like in the system architecture from Bronkers et al. (2011), shown in Figure 2.7, the C2 information system communicates its orders in C-BML to HLA. Our Command Agents pick up the C-BML order, process it and output a number of Low-level BML orders over HLA to (the plugin of) the simulator. Reports, including task reports, locations of units and data for our MOEs, are sent back from the simulator to the agents and eventually the C2 system.

New to our system architecture is the additional standard in use: MSDL, see Section 2.2.1.2. The MSDL file is created from the scenario in the C2 workstation and then used as input for the Command Agents and the simulator, synchronizing all systems. Furthermore, the new decision support system is added with a connection to our multi-agent system, which provides it with its data. The only human in the loop is the commander working on the C2 workstation, who can now also access the DSS for a quick overview of simulated results.

In the future, the C-BML orders and reports and the MSDL file will be transferred through web-services, which are provided in MSG-085, instead of the direct gateway connection.

## 4.3 Conceptual Design - Agent

In this section the updates to the Command Agents from Figure 4.6 will be discussed. First the architecture of the multi-agent system will be explained, followed by descriptions of the updates that have been made to the agents.

### 4.3.1 Agent Architecture

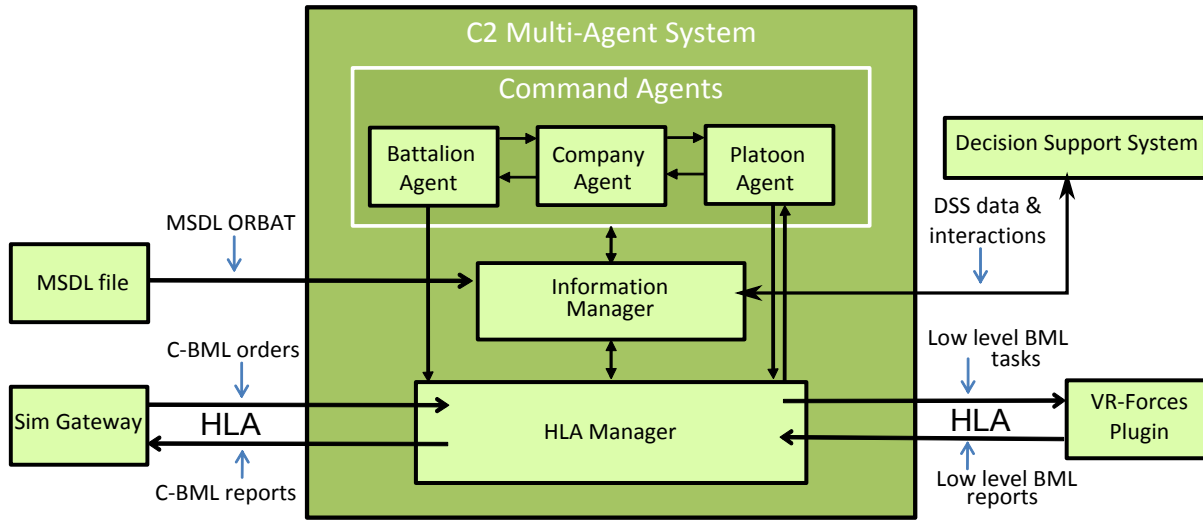


Figure 4.7: Agent architecture

Our agent architecture started out as shown in Figure 2.8, created by Bronkers (2011). The general description of these agents can be found in Section 2.2.4.2. Subsequently, we adapted the agent architecture to incorporate the decision support system, as can be seen in Figure 4.7. The other change that has been made to this architecture is that there is only one agent between the Command Agents and the HLA Manager, the new Information Manager. This Information Manager has replaced (most of) the Stub agent and the StartAgents agent, next to its new capabilities. In the following sections, only the differences from Bronkers' agents will be discussed, not each agent's entire BDI systems.

### 4.3.2 HLA Manager

The HLA Manager agent, described in Section 2.2.4.2, has been adjusted to provide the Command Agents with more information from VR-Forces so that they could act more intelligently. Previously, the HLA Manager only provided *GroundVehicleObjects*, *AggregateEntityObjects* and *TaskReportEvents* to the agents<sup>3</sup>. Now, *GroundVehicleObject* has been extended to its superclass *PhysicalEntityObject*, allowing a.o. humans to be received from HLA as well. Internally, every *PhysicalEntityObject* is transformed into an *Equipment*<sup>4</sup> object, a self-made java class which will be used throughout the multi-agent system when referring to single units.

Furthermore, the HLA Manager's communication with HLA has been extended to support the new required behaviour and the new DSS. See Appendix E for more information on this extension in the (HLA) FOM and Section 5.1 for implementation in the simulator's plugin.

<sup>3</sup>These object are HLA RPR-FOM standards, see Section 2.2.1.3

<sup>4</sup>Equipment is the type provided in MSDL for tanks and soldiers. Since we initiate the agents from MSDL now, we kept the name.

### 4.3.3 Information Manager

The Information Manager is a new agent that incorporates all abilities of the previous StartAgents agent and Stub agent from Bronkers (2011), which can be found in Section 2.2.4.2. The Information Manager is able to initialize the correct agents based on a provided ORBAT in an accompanying MSDL file, instead of having the numbers and types of agents to be started hard-coded in the StartAgents plan. For example, if the ORBAT is made up of 1 Battalion with 3 Companies, which each have 3 Platoons, the Information Manager will start 1 Battalion agent, 3 Company agents and 12 Platoon agents, and connect the subordinates to their superiors.

Subsequently, the Information Manager stores the ORBAT in its beliefs by creating a new OrbatMap (java class) in which Aggregates and Equipments are stored in a tree of HashMaps, as the military hierarchy dictates. The OrbatMap allows storage of new Aggregates (agents) and Equipments, retrieval and updating of stored Aggregates and Equipments. Therefore, updates from the simulator are now sent from the HLA Manager to the Information Manager instead of to all Platoon agents. The Information Manager then updates the OrbatMap accordingly, thereby supporting a single centralized storage. When an Equipment is updated, its superior Platoon agent will be informed about this update. When an Aggregate is updated, the corresponding agent will be informed about this update. By centralizing this management of information, it only has to be stored in one place, taking less memory space and the information will always be synchronized between the agents. The downside to the centralization of information is that new communication lines will have to be added to retrieve this information from the Information Manager any time one of the other agents wants to reason on this information.

### 4.3.4 Stub agent

Even though the Stub agent is no longer strictly part of the agent architecture, it is still in use in certain situations. It used to receive the BML orders from the HLA Manager before it was sent to the Battalion agent, but this task has been taken over by the Information Manager. Therefore, the stub agent has been excluded from the multi-agent system when a C2 information system (C2IS) is attached. Only when the MAS and simulator are connected without C2IS, e.g. during development of the agent, the Stub agent is used to act as a proxy, allowing test orders to be sent into HLA.

### 4.3.5 Battalion agent

The Battalion agent has been updated to respond and take the decisions for the CIS *Hasty occupation of a blocking position*. It will decide whether a blocking position should be ordered based on enemy behaviour, and what company will have to block this enemy. Other CISs do not require input from the Battalion agent, so this is the only coordination on company level that is used in this project.

### 4.3.6 Company agent

The Company agent has been extended the most with the additions of the CISs. All CISs require coordination between platoons, and this coordination is to be handled on the Company

agent level. The Company agent is now able to:

- coordinate the platoons into the different phases of the assault at the same time;
- translate the assault phase behaviour to platoon behaviour;
- interrupt the assault for the hasty occupation of a blocking position, when ordered to;
- continue with the assault at the correct phase, when ordered to;
- choose the formations to be used;
- coordinate the platoons into formations (the simulator is not tasked on the company level, so the company agent has to enforce & create the formations);

#### 4.3.7 Platoon agent

The changes for the platoon agent are in its in- and output. Now it receives new tasks like MoveIntoFormation to forward to the HLA Manager, where previously it only processed Attack, Defend and Move tasks. However, the platoon agent still needs not reason on these tasks because they can directly be forwarded to the aggregate in VR-Forces. The reasoning on what tasks to sent to VR-Forces is done primarily on the Company agent level and coordination of tanks is done by the simulator.

Also, as discussed earlier, the platoon agent does not receive updates for all Equipment anymore, these are only sent to the Information Manager. One reason for this change was that it was not possible to only receive the subordinate Equipment; all equipment would be received. Now that it is centralized, the Information Manager can send the subordinate Equipments to their superior Platoon, keeping all data up to date, while not storing every Equipment in every single Platoon agent. The platoon will also receive the status of its Aggregate counterpart anytime the Information Manager notices a change to it.

### 4.4 Conceptual Design - Decision Support System (DSS)

The decision support system (DSS) should be a system which will help the military commander during the comparison and evaluation of his course of actions (COAs). We have researched requirements for a DSS in the command & control domain from related literature and we have discussed with the military experts about their vision for such a system. The following sections will discuss the concepts we started with.

#### 4.4.1 Initial concept

See Figure 4.8 for the initial sketch and idea for the DSS. Two COAs are given on the C2IS, for example one longer route and one shorter route, but with expected enemy resistance. The DSS would have a row for each COA and columns for the different MOEs, while allowing more MOEs to be added (as columns) and more COAs to be added as rows. There might also be an option to add a computer generated alternative route to compare to the user supplied COAs, but we

have already chosen to focus our attention on doctrine instead. After running the simulation, the DSS table would then be filled with values for all these COAs and MOEs, allowing the commander a quick overview of the expected effects the candidate COAs.

#### 4.4.2 Measures of Effectiveness (MOEs)

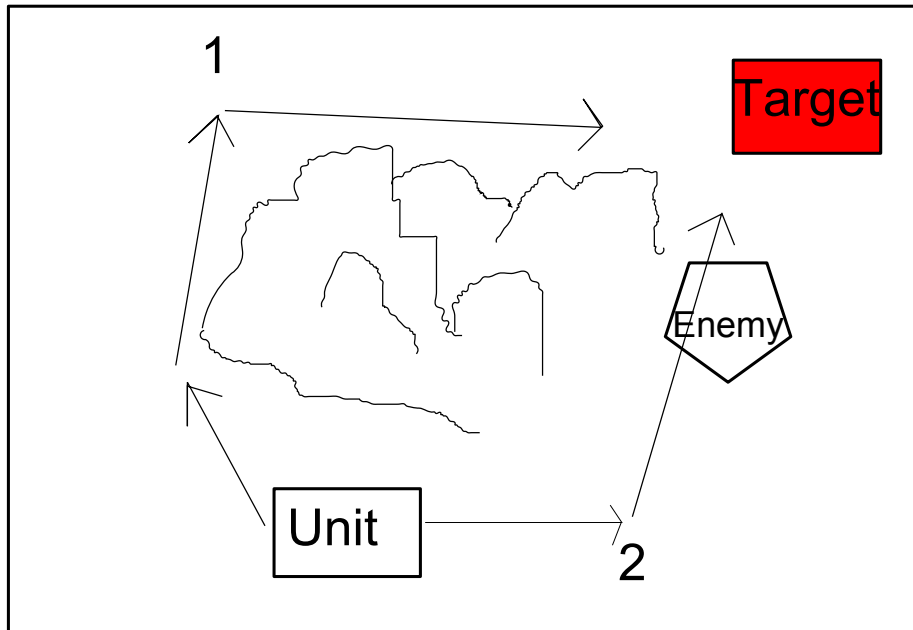
The DSS is to be used to compare user provided COAs. These COAs will need to be compared on certain measures of effectiveness (MOEs), or criteria, based on the events that transpired in the simulator or agents. In the brainstorm session on November 9th, see Section 3.1, we came up with the following possible criteria for comparison of COAs:

- Time
- Survival Rate
- Fuel usage
- Ammunition Usage
- Enemies destroyed
- Enemies encountered
- Weather
- Bridges
- Roads
- Off-road
- Forests
- Alternative / Fallback options

Time is the time taken for completing the COA; survival rate is the percentage of friendly units that have survived; fuel usage is the percentage of fuel left; ammunition usage is the percentage of ammunition left; enemies destroyed and encountered is similar to survival rate but for the enemy units; weather is rain or sun or fog, which can influence the operation; bridges is the number of them that have to be crossed, since bridges are chokepoints; roads are an easy and fast mode of transport; off-road is a more difficult and costly mode of transport, thus road and off-road comparison might favor a COA; forests make for good cover yet provide smaller line of sight, so the meters of forest crossed might impact a COA; Alternative or fallback options in a COA would show how versatile it is, e.g. if there would be multiple directions from which to attack. This set of MOEs is still only a subset of the criteria for COA analysis and comparison.



### Compare Routes



|   | MoE1 | MoE2 | MoE3 | MoE4 | MoE5 | +ADD MoE | MoE6 |
|---|------|------|------|------|------|----------|------|
| 1 | ...  | ...  | ...  | ...  | ...  |          | MoE7 |
| 2 | ...  | ...  | ...  | ...  | ...  |          | ...  |
|   |      |      |      |      |      |          |      |

+ADD  
New Route

+GENERATE  
Alternative Route

**Figure 4.8:** Concept of the DSS: C2IS shown above, with 2 routes towards a target. DSS shown below, with 2 rows representing the routes and columns filled with data retrieved from the simulator.

Chapter 5

Implementation

In this chapter we shall describe what part of our model has actually been implemented and how we accomplished that. We shall start with the new communication additions that have been made towards VR-Forces, in Section 5.1, and to C2 information system in Section 5.2. Thereafter, we shall discuss the implementation of all the CISs in the agents in Section 5.3. Finally, the chapter shall finish with the implementation of the DSS and its MOEs in Section 5.4.

As discussed in Section 4.1.2, we have chosen 3 main CISs to be implemented:

1. Assault an enemy position (or "Assault")
2. Tactical Road March ("March")
3. Hasty occupation of a blocking position ("Blocking Position")

All of these, and the Line and Column formations, have been implemented. Furthermore, we have been able to input the following MOEs to our decision support system:

- Friendly survival rate
- Fuel usage
- Enemy survival rate

## 5.1 Communicating to VR-Forces: New in- & output requirements

We need extra in- and output from VR-Forces for the correct implementation of CISs and for the measurement of the MOEs. These input and output requirements were not yet available for the agents from the plugin to VR-Forces. All information available from the plugin to VR-Forces to the agents is provided via HLA and the specification of what can be sent over this link is written in an ontology called the *FOM*, see Section 2.2.1.3 for more information, currently shared by TNO and FFI. So, we needed to add our new interactions to this FOM, after which it was possible to generate code based on this FOM for the plugin. After generation, we had to adjust parts of this code, on the VR-Forces plugin side, to get the right information from VR-Forces into these interaction messages. The plugin allows for the low-level BML sent by the agent to be acted upon in the simulator, and the needed information to be sent back to the agent. The generated code sets up the sending and receiving mechanisms and parameters, as is described in the FOM, but it is up to the engineer to get the correct information into and from these messages. Some of these requirements were already available in VR-Forces, so a 1-on-1 connection could be created between the message and the information or action from VR-Forces. However, others would require a lot more work; if they were not vital to the success of this project, they were discarded. In Appendix E, you can find the new FOM and the table of changes it has undergone.

The following list shows what was required ("R") or wanted ("W") from VR-Forces for the CISs and MOEs to be implemented successfully. Wanted requirements were not vital to the successful implementation of the CISs, and can generally be hard-coded or otherwise assumed in the agents. The CIS or MOE for which it is required or wanted is also mentioned.

### Output to simulator

- Set unit's heading  
(R; CIS: Block)
- Set unit's speed  
(R; CIS: Assault, Block, March)
- Abort current order  
(R; CIS: Block, Assault)
- Move into formation: Line, Column  
(W; CIS: Assault, March)  
or, alternatively, Follow unit with distance X,Y  
(W; CIS: March)

### Input from simulator

- Time elapsed  
(R; CIS: Assault; MOE: Time)
- Unit's amount of fuel  
(R; MOE: fuel usage)
- Unit's amount of ammunition  
(R; MOE: ammunition usage)
- Terrain type currently driving on  
(W; CIS: March; MOE: Roads, Off-road, Bridges)
- Weather Conditions  
(W; CIS: March)
- Nearby forest areas  
(W; MOE: forests)

The following is the list of what has been implemented in the end:

### Output

- Set unit's heading
- Set unit's speed
- Abort currently executing order
- Move into formation: Line, Column
- Follow unit with distance X,Y,Z

### Input

- Entity's amount of fuel
- Entity's amount of ammunition

In the following sections we shall describe how all of these have been implemented.

#### 5.1.1 Set Heading, Turn To Heading & Set Speed

Set Heading and Set Speed are both a set, which means that they take effect immediately, unlike Turn To Heading, a task, which has to be simulated. However, these three are all related since their implementation is mostly similar. We needed these to set speed of our traveling formations and the heading of formations. In Code 5.1, 5.2 and 5.3, you can see excerpts of the definitions from the FOM. The name of the interaction is provided together with a parameter that the agents have to attach: the heading or speed. The superior classes in the FOM already provide ways for the agents to provide the target object who's heading or speeds is to be adjusted.

**Code 5.1:** XML code from the FOM for Set Speed

```

1 <interactionClass>
2   <name>SetSpeed </name>
3   ...
4   <semantics>Change the
      ordered speed of
      a unit.
   </semantics>
5   <parameter>
6     <name>Speed </name>
7     <dataType>HLAfloat ...
8     <semantics>Determines
      the ordered
      speed to set
      for a unit, in
      m/s.
   </semantics>
9   </parameter>
10 </interactionClass>

```

**Code 5.2:** XML code from the FOM for Set Heading

```

1 <interactionClass>
2   <name>SetHeading
      </name>
3   ...
4   <semantics>Change the
      heading of a
      unit. </semantics>
5   <parameter>
6     <name>Heading
      </name>
7     <dataType>HLAfloat ...
8     <semantics>Determines
      the heading to
      set for a unit,
      in radians.
   </semantics>
9   </parameter>
10 </interactionClass>

```

**Code 5.3:** XML code from the FOM for Turn To Heading

```

1 <interactionClass>
2   <name>TurnToHeading
      </name>
3   ...
4   <semantics>Tasks a
      unit to turn to
      the given
      heading.
   </semantics>
5   <parameter>
6     <name>Heading
      </name>
7     <dataType>HLAfloat ...
8     <semantics>The
      heading for the
      unit to turn
      to, in radians.
   </semantics>
9   </parameter>
10 </interactionClass>

```

In the VR-Forces GUI, one can manually task the units. By matching what tasks are already possible in the GUI to the behaviour we want, we get an easier implementation of our behaviour in the simulator's plugin. For example, the implementation for the Set Heading is given in Code 5.4. This method is called whenever the plugin receives a SetHeading interaction. It then just retrieves the entity that is to be adjusted, creates a VR-Forces'-provided SetHeading request to which the interaction's parameter (the heading) is added. Finally the set is processed in the simulator with VR-Forces' sendSetDataRequest() method. The SetSpeed interaction is handled in the same way, using a SetSpeedRequest. The TurnToHeading interaction is handled a little differently, since it is a task instead of a set. However, basically VR-Forces' TurnToHeadingTask is retrieved and the parameter (heading) from the interaction is added to this task. Finally this task is (planned to be) executed by the entity by wrapping the task in a TaskMessage and tasking the entity with that task. For every task, an ID is stored as well, which will be used to send a report to our agents when the task is tasked or completed later on.

**Code 5.4:** C++ code for executing Set Heading in VR-Forces plugin

```

1 void receiveSetHeading(BML::DtSetHeadingInteraction* inter, void* usr){
2   DtVrfObject* entity = getCommandedObject(inter);
3
4   DtSetHeadingRequest request;
5   DtReal heading = (DtReal) inter->heading();
6   request.setHeading(heading);
7
8   myCgf->sendSetDataRequest(entity, request);
9 }

```

### 5.1.2 Abort executing task

Abort executing task is a task that we needed to stop tasks that the simulator is executing. Previously every new task from the agent would have to wait until the current executing task was completed. However, our reactive blocking position possibly requires a simulated company to interrupt its current task, which is now possible due to Abort executing task, see Code 5.5. Just as the FOM addition is very simple, essentially only a name, the implementation of AbortExecutingTask is simple too: every entity has a taskManager in VR-Forces and this manager allows the skipTask() operation, causing the entity to stop executing its current task.

**Code 5.5:** XML code from the FOM for AbortExecutingTask

```

1 <interactionClass>
2   <name>AbortExecutingTask</name>
3   ...
4   <semantics>Stop the task the unit is currently executing.</semantics>
5 </interactionClass>

```

### 5.1.3 Move into formation

Move into formation is the biggest addition to the agent's new behaviour and is used in every one of our implemented CISs. Luckily this formation function was already available in VR-Forces, so once again we just mimicked the parameters needed in the VR-Forces GUI, see Code 5.6. This task consists of a formation reference, which is just a name like Line; coordinates of the location where to initialize this formation; and, the heading of the formation.

**Code 5.6:** XML code from the FOM for MoveIntoFormation

```

1 <interactionClass>
2   <name>MoveIntoFormation</name>
3   ...
4   <semantics>Tasks an aggregate unit to move into the given formation with the
      given heading.</semantics>
5   <parameter>
6     <name>Formation</name>
7     <dataType>HLAASCIIstring</dataType>
8     <semantics>The category of positional arrangement of the entities within the
      aggregate.</semantics>
9   </parameter>
10  <parameter>
11    <name>Location</name>
12    <dataType>WorldLocationStruct</dataType>
13    <semantics>The location (x, y and z) to move to.</semantics>
14  </parameter>
15  <parameter>
16    <name>Heading</name>
17    <dataType>HLAfloat...</dataType>
18    <semantics>The heading of the formation in radians.</semantics>
19  </parameter>
20 </interactionClass>

```

The implementation is once again relatively straight-forward, as can be seen in Code 5.7. The interaction that triggers this method provides it with the previously discussed parameters, which can be added to VR-Forces' MoveIntoFormationTask. The only change made to the parameters is that the received location is clamped to the ground, i.e. the height value is automatically retrieved from the simulator. Eventually the task is executed by the tasked entity, causing the simulator to move it into formation at the provided location and heading. The formation is a name that is specific to the simulated entity: each M1A2 tank and M1A2 tank aggregate (platoon / company) has a configuration file used by the simulator to check what that formation name refers to. We have adjusted these configuration and formation files so that we could task them with the formations from our CISs, by providing a simple name such as *CIS\_Line*.

**Code 5.7:** C++ code for executing MoveIntoFormation in VR-Forces plugin

```

1 void receiveMoveIntoFormation(BML::DtMoveIntoFormationInteraction* inter, void*
  usr){
2   DtVrfObject* entity = getCommandedObject(inter);
3
4   // Create the task
5   DtMoveIntoFormationTask moveIntoFormationTask;
6
7   // Clamp the geocentric position from the interaction to the terrain
8   // First convert it to the local position, then change the z-value and finally
   convert it back to
9   // a geocentric clampedPosition that can be used as the location.
10  DtVector position = DtVector(inter->location().x(), inter->location().y(),
   inter->location().z());
11  DtVector clampedPosition;
12  ...
13  moveIntoFormationTask.setLocation(clampedPosition);
14
15  // Set the formation heading as provided
16  moveIntoFormationTask.setHeading(inter->heading());
17
18  // Set the formation name as provided
19  ...
20  moveIntoFormationTask.setFormationName(formation);
21
22  // Wrap the task in a task message and initialize the task message
23  ...
24
25  // Execute the task
26  entity->taskManager()->executeTask(taskMsg);
27
28  // Retrieve and store the VR-Forces task ID for the assigned task
29  ...
30 }

```

#### 5.1.4 Follow

The follow task is added to the FOM as FollowEntity, a task native to VR-Forces as well. Follow was wanted to implement formations of the company, by ordering platoons to follow each other at specified distances. This task requires four parameters: AffectedWho, the entity to be followed; Behind; Right; and Above. The latter three are the distances at which to follow the affected entity. Eventually we implemented this similar to MoveIntoFormation, however it turned out that in VR-Forces an aggregate can not follow an aggregate. In other words, we can not task a platoon or company to follow another platoon or company. Add to this the fact that we only task the simulator on the platoon (aggregate) level, thus we could not use follow.

#### 5.1.5 EntityFuel & EntityAmmunition

EntityFuel and EntityAmmunition are interactions that VR-Forces sends to the agents, after the agents have registered to receive them. They are used to provide the agents with information about the fuel and ammunition levels of an entity in the simulator, which the agent can eventually turn into a MOE for our DSS. Most information regarding entities is received in one entity object with some parameters like a name, ID, speed, heading. However, none of the available objects included fuel or ammunition levels, and it was not possible for us to adjust these objects, only to add new interactions that will be sent. As such, we created these two interactions, which are very similar, see Code 5.8 for entityFuel. As can be seen, VR-Forces will report to the agents the type of fuel and the percentage it is at for this entity. For ammunition this is the same.

**Code 5.8:** XML code from the FOM for EntityFuel

```

1 <interactionClass>
2   <name>EntityFuel</name>
3   ...
4   <semantics>Represents a report from an unit about the amount of fuel it has
      left.</semantics>
5   <parameter>
6     <name>FuelType</name>
7     <dataType>HLAASCIIstring</dataType>
8     <semantics>The type of fuel</semantics>
9   </parameter>
10  <parameter>
11    <name>FuelPercentage</name>
12    <dataType>HLAfloat32BE_perc_perfectalways</dataType>
13    <semantics>The amount of fuel the entity has not yet used, in
      percentages.</semantics>
14  </parameter>
15 </interactionClass>

```

The implementation of these interactions concerns itself with triggering these reports, which in VR-Forces is not trivial. When do we want the simulator to send these reports? Every millisecond? When asked? When the value of fuel or ammunition changes? The latter seemed most useful to us, but such a trigger did not exist. Instead, we found the possibility in VR-Forces to create a *Predicate*, which is a condition trigger that allows us to compare the current value



of a resource, such as fuel, to a predefined value. As such, we implemented our desired trigger, by creating such Predicates for a set of percentages. As a result, this Predicate now triggers when the value of our desired resource drops below a certain set of threshold percentages, and then a report is sent back to our agents using the code in Code 5.9. In the first block of code, parameters from the Predicate are retrieved, such as what threshold is currently triggered. Then in the second block of code, starting at *resourcePercentage->find(...)*, line 10, we check if we already sent a report for this threshold, since a value of 85% fuel is always going to be below 90% and thus trigger that Predicate while we only want to receive this report once. Finally, if we did not send a report yet, we send the report to our agent.

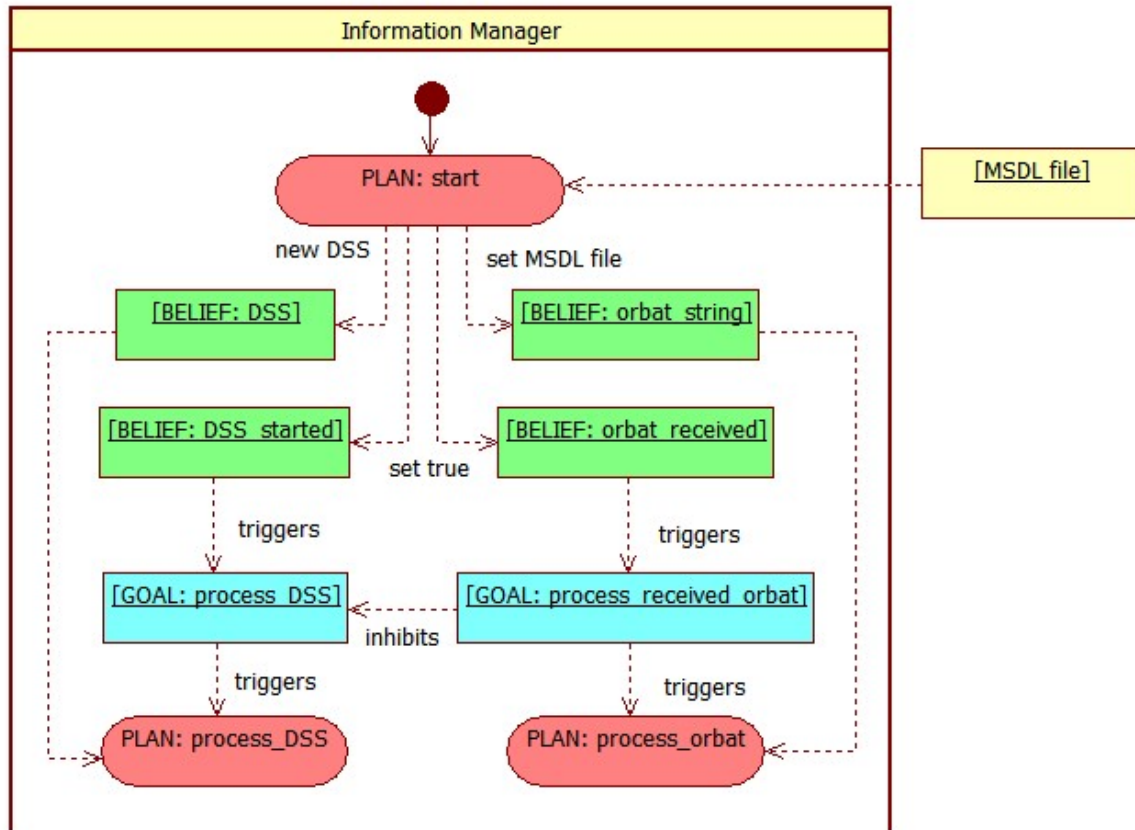
**Code 5.9:** C++ code for sending *EntityFuel* reports to our agents

```

1 void entityFuelCallback(DtVrfObject *obj, DtVrfObjectPredicateEvaluator
  *predicateEval, void *userData){
2
3   IntToBoolMap* resourcePercentage = static_cast<IntToBoolMap*>(userData);
4   BmlEntityFuelPredicate* predicate = (BmlEntityFuelPredicate*)
      predicateEval->predicate();
5   DtString fuelType = predicate->resource();
6   DtReal fuelPercentage = predicate->comparisonValue();
7   DtString fuelOper = predicate->oper();
8   bool fuelPercBool = predicate->percentFlag();
9
10  IntToBoolMap::iterator iter = resourcePercentage->find(fuelPercentage); //find
      the comparisonValue in userData
11  if (iter != resourcePercentage->end()) {
12    // Found the percentage. Check stored state
13    if (iter->second) {
14      // If the boolean = true, we have already sent this report.
15    } else {
16      // How did it get added to the userData without true boolean?
17      obj->objectConsoleWarn() << obj->objectName() << "_has_" << fuelType << "_
        below_" << fuelPercentage << ". It was already on the percentage list?"
        << std::endl;
18      sendEntityFuelReport(obj, fuelType, fuelPercentage); // threshold
        percentage reached
19      // Store the comparisonvalue
20      resourcePercentage->insert(IntBoolPair(fuelPercentage, true));
21    }
22  } else {
23    // The percentage was not stored in the userData yet
24    obj->objectConsoleWarn() << obj->objectName() << "_has_" << fuelType << "_
        below_" << fuelPercentage << std::endl;
25    sendEntityFuelReport(obj, fuelType, fuelPercentage); // threshold percentage
        reached
26    // Store the comparisonvalue
27    resourcePercentage->insert(IntBoolPair(fuelPercentage, true));
28  }
29 }

```

## 5.2 Communicating with C2IS: Processing the MSDL ORBAT



**Figure 5.1:** *The BDI reasoning after starting the Information Manager agent, leading to the start of the other agents from MSDL and the creation of the DSS afterwards.*

On the other end of our system architecture, Figure 4.6, we have the new communication with the C2 information system in the form of the MSDL file, now used to create the correct agents. The processing of the MSDL file is done by the Information Manager after it has been initiated, see Figure 5.1 for the BDI representation of its reasoning. It starts with the triggering of the start plan, which sets up the values for four different beliefs: DSS, in which a newly created DSS is stored; orbat\_string, in which the MSDL file is stored as a string; and DSS\_started and orbat\_received, which are booleans set to true. The booleans trigger the creation of new goals process\_DSS and process\_received\_orbat in the agent. The process\_received\_orbat goal is acted on first since it inhibits the process\_DSS goal, which will be acted upon only if process\_received\_orbat is not active anymore. At this point, the process\_orbat plan is triggered, which will take the orbat\_string from the agent's beliefs, decypher the XML with Java's JAXB<sup>1</sup>

<sup>1</sup>JAXB: <http://www.oracle.com/technetwork/articles/javase/index-140168.html>

and then retrieve useful information from it. Consult Appendix D for more information about the XML structure of an MSDL file. In the MSDL file we can find Units and Equipment, where Units represent groups of entities, such as a company or platoon. The Units that belong to the agent force are then turned into new agents and Aggregate objects. The Aggregate object will be updated from VR-Forces and is subsequently needed to communicate with VR-Forces. Equipment hold our actual entities, the tanks, for which no agents are created. Instead, they are turned into Equipment objects, which will also be updated from and communicated to VR-Forces. Once the whole MSDL file is processed, the `process_received_orbat` goal has succeeded and the agent will continue with initializing the DSS.

### 5.3 Agent Behaviour: Implementation of the CISs

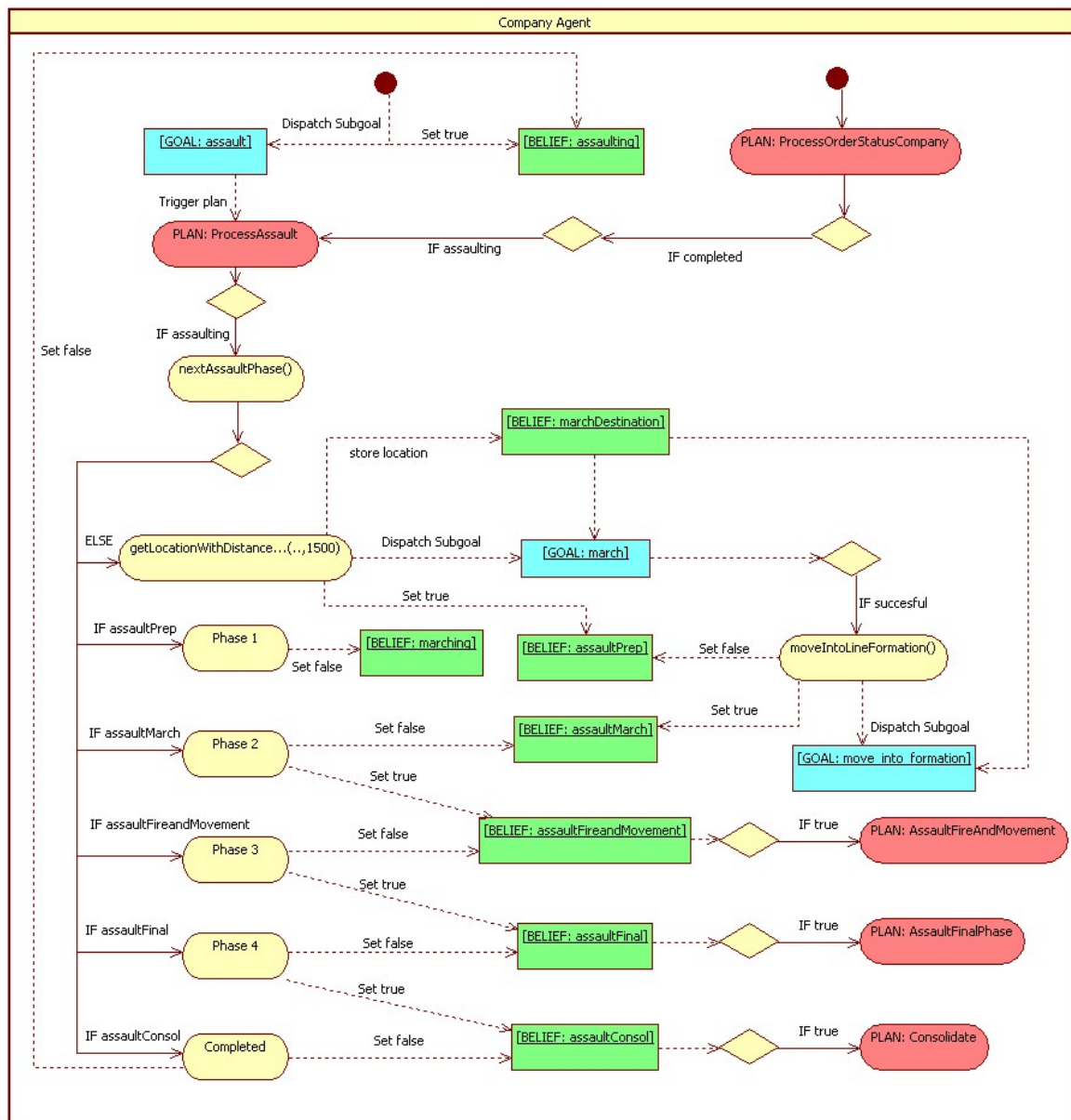
In this section of the implementation chapter we shall discuss the main part of our implementation: the combat instruction sets. We shall start with the Assault CIS in Section 5.3.1, followed by Tactical Road March in Section 5.3.2, the Blocking Position in Section 5.3.3 and finally the Line and Column formations in Section 5.3.4. For each CIS, we will provide a short rehash before we go into details of implementation.

#### 5.3.1 CIS: Assault on Enemy Position

The assault on an enemy position consists of several phases, and these have been implemented as well. Once tasked with the assault, the company agent will create the assault goal, triggering the *ProcessAssault* plan. This plan will keep track of the phases and start up the assault. By building the assault in phases, we create a modular design, in which parts can be more easily replaced if needed. Furthermore, more in-depth implementations of particular phases can later on be added while the overall behaviour already works. But most important of all, by keeping track of the phases, the agent will be able to interrupt and continue the assault later on without having to start all over. See Figure 5.2 for the BDI approach taken by the Company agent to select the correct part of the Assault. The Assault will start when the *assault* goal is dispatched (created) and the *assaulting* belief is set to true. As long as the assaulting belief is true, the agent will act in the context of assaulting. As such, whenever it receives a completed task report in the *ProcessOrderStatusCompany* plan, only while it is assaulting, it will trigger the *ProcessAssault* plan, which takes care of setting the correct phase.

##### 5.3.1.1 Phase 1 : Preparation

Phase 1 consists of the march towards the start-line, so it starts with the computation of the start-line location. The location taken as the start-line is the closest location that is 1500 meters away from the target location (of the assault). Afterwards, a *March* goal is dispatched for the start-line location and the belief *assaultPrep* is set to true, allowing the agent to remember that this is the current assault phase. Once a successful task status report is received, assumed to be the march, the march is triggered once more by changing a belief. This is required because the march consists of two parts. When the march to the start-line is completed, the *assaultPrep* belief is turned to false, the *assaultMarch* belief is turned to true (signifying the success of the



**Figure 5.2:** The BDI processing of the assault phases in the Company agent. The blue blocks are goals, the green blocks are beliefs and the red ovals are plans. The beige ovals are actions (intentions) in plans.

march) and a *move\_into\_formation* goal is dispatched for line formation at the start-line. The next time the ProcessAssault plan is called upon, which is after every successful task status report while assaulting, the next phase of the assault is initialized by setting assaultMarch to false and *assaultFireandMovement* to true.

#### 5.3.1.2 Phase 2 : Fire and Movement

Once the company agent's belief assaultFireandMovement is set to true, a new plan is triggered: the assaultFireAndMovement plan. In this plan, the final assault line, 600 meters from the target location, is computed in the same way the start-line was computed. Afterwards, the company agent dispatches a new *move\_into\_formation* goal for a line formation at this final assault line. Furthermore, the agent sets the rules of engagement (ROE) to fire when fired upon. This will simulate the defensive movement from the start-line to the final line.

**Bounds** The CIS states that, if fired upon during this phase, movement should be changed from one line to bounding lines. However this has not been implemented during this project. The detection of fire would have to be added and then this could be used in a belief to trigger a new bounding fire and movement plan, which would abort the current action and impose its own way of movement.

**Completion** A successful task status report from this move into formation will trigger the initial ProcessAssault plan once again. This time, assaultFireandMovement is set to false while *assaultFinal* is set to true.

#### 5.3.1.3 Phase 3 : Final attack

The positive assaultFinal belief will trigger the *assaultFinalPhase* plan. This plan will set the speed to maximum, the ROE to fire at will and dispatch a *move\_into\_formation* goal for a line formation at the assault's target location. Similar to the previously implemented attack (Bronkers, 2011) , this will simulate the tanks attacking the enemy position, yet this time in an orderly coordinated line with maximum fire power in front. As was previously the case, the completion of this task will cause ProcessAssault to trigger again. This time, the belief assaultFinal is set to false and the belief *assaultConsol* is set to true.

#### 5.3.1.4 Phase 4 : Consolidate and Reorganize

The company agent's belief assaultConsol will trigger the Consolidate plan. This plan will compute a location with 400 meters distance from the assault's target location, set the ROE to fire when fired upon and dispatch the goal to setup battle position (with 500 meters frontage per platoon) formations at that location.

#### 5.3.1.5 Finalizing the assault

If consolidate returns a successful task status report, the ProcessAssault plan will be triggered for the final time. At this point in time, it will clear the agent's beliefs of all those related to the

execution of the assault task. This will cause future completeds to not trigger the ProcessAssault plan anymore.

### 5.3.2 CIS: Tactical Road March

The Tactical Road March is a march along roads towards the assembly area, but our agents do not receive road information from the simulator. However, the agents assume that the march is conducted along a road. In future work, this might also be implemented. We have implemented the Tactical Road March as a move in formation towards a destination. As a counterpart, we have also created an off-road march. When ordered, or desired, the company agent will create a March goal, to travel to a destination either on- or off-road at march speed. The TacticalRoadMarch plan can then be used to execute this goal.

#### 5.3.2.1 TacticalRoadMarch plan

The company agent's TacticalRoadMarch plan dispatches a `move_into_formation` goal for a road march formation at the agent's current location. The road march formation has been stored as a `.frm` file for use by the agents and simulator, see also Figures 4.2a and 4.2b. Before dispatching the goal, the agent creates a marching belief, a Boolean set to true. If the agent receives confirmation from the simulator that the formation has been achieved at the desired location, it sets the marching belief to false, thereby triggering the TacticalRoadMarch plan again. This time, the plan dispatches its final `move_into_formation` goal for a road march formation at the destination, and turns marching back on. When processed, this will cause the simulator to move the platoon aggregates in their road march formations to the destination. When marching is once again turned to false, the plan and the March goal that triggered it succeed.

This use of the marching belief is a work-around that is needed because of the delay in executing a plan and receiving the result of the plan. In Jadex, a `performgoal` (which all our goals are), like the tactical road march and the move into formation, will be successful if a plan has successfully been employed on its behalf. As such, when we dispatch a move into formation goal in the TacticalRoadMarch plan for the first part of the plan, this part will have succeeded when the company agent has ordered its platoon agents to move into formation. At that point, it will want to carry on with the TacticalRoadMarch plan and dispatch the second move into formation goal (for moving from the starting point to the destination). However, there probably has not been any action in the simulator yet at this point because the platoon agents are still processing their move into formation plans. Therefore, the move into formation would already overwrite the previous while that has not yet been executed or completed in the simulator. This is a definite misconnection between our system architecture and the jadex framework; our agents do not receive immediate responses from their plans. So instead we had the work-around belief that has been used to make the TacticalRoadMarch plan wait for a completed task report first, before it moves onto step 2.

### 5.3.2.2 OffRoadMarch plan

The off-road march is exactly the same as the Tactical Road March, except that the formation to be moved in is set to a column formation.

## 5.3.3 CIS: Hasty Occupation of a Blocking Position

### 5.3.3.1 Distance threshold

The hasty occupation of a blocking position is a CIS reacting to unforeseen enemy progress, hence our agents also need to activate this plan as a reaction to such an observation. To represent not foreseen enemy progress, we decided to monitor expected enemy locations and the distance from these expected locations of observed enemy equipment in our battalion agent. If the enemy equipment's distance from expected locations is greater than some threshold, it is considered to show unforeseen enemy progress, and it should be reacted to with the hasty occupation of a blocking position from the nearest company agent. While a lot more reasoning could be used before a blocking position is applied, this distance threshold was chosen as it seemed to be the easiest and quickest implementation, while still showing the reactive capabilities that these agents can provide.

### 5.3.3.2 Expected locations

For our expected locations, we chose the positions from orders previously provided to the agents. When the order to assault a location is given, we can expect there to be enemy activity around this location. Furthermore, since the order to apply a blocking position now also records that location-to-be-blocked as an expected location, the distance threshold automatically ensures that the blocking position is not ordered numerous times for the same entity. If this was not the case, every time the update of the enemy's location is provided to the agents, if it still passes the distance threshold, a new blocking position would be ordered and executed. To fine-tune this reactive behaviour, at the start of a scenario, certain locations could already be provided to the agents as expected locations. Moreover, the distance threshold could be adjusted beforehand.

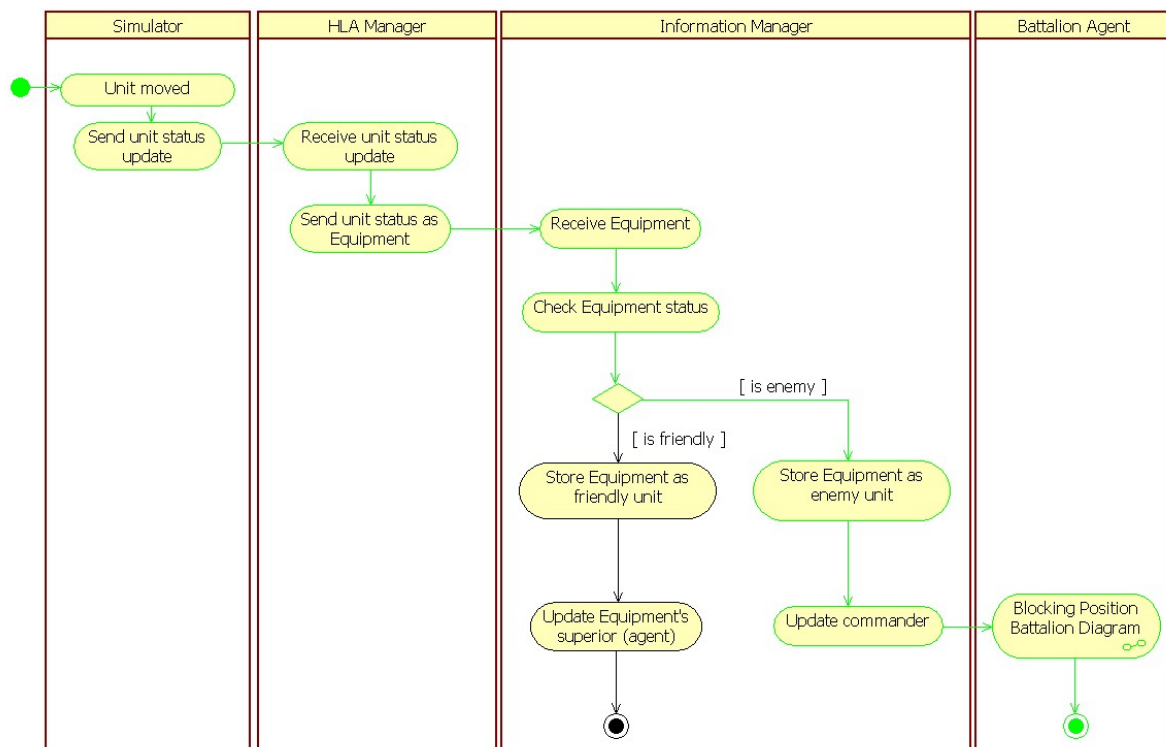
### 5.3.3.3 Blocking positions

The positions of the platoons and their tanks during a blocking position ordered to the superior company has been provided to the agents and the simulator as a formation. The CIS provides some default distance values for this formation: the frontage of a platoon measures 500 meters; the distance between two platoons measures 100 meters from side to side; and the equipment of a platoon are equally distributed over this frontage in a line-like formation, as can be seen in Figure 4.5b. As such, the distance between our platoons (aggregates) is 600 meters, as measured from their centres of gravity. The platoon's line formation is extended to a 500 meters length, thus providing a horizontal distance of 166 meters between the every other platoon. Once ordered to occupy the blocking position by its superior battalion agent, the company will order its platoons to set-up this formation, facing towards the position-to-be-blocked. This reuses the heavily used move into formation task that the agents can send to the simulator. Furthermore,

each entity's rules of engagement shall be set to fire at will, so that the threat is eliminated as soon as possible.

#### 5.3.3.4 Suspension of other tasks

The ordered company agent might already be executing a task at the time of the reactive blocking positions. Normally, new tasks are put on hold until the current task has been completed. To circumvent this structure, the abort executing task task has been added to the agents' repertoires. As such, the superior battalion will first send the abort executing task task. This will abort execution of the current task and suspend tasks that were put on hold. However, because we might not want to discard these tasks entirely, we have introduced the continue task as well. Effectively negating the abort executing task tasks sent before, the agent is ordered to continue with any suspended or aborted tasks.



**Figure 5.3:** An activity diagram showing the events leading up to a blocking position. The green path shows how an update from the simulator of a formerly unknown enemy equipment causes the Information Manager to Update Commander (Battalion Agent) about the new unit. The black path shows what happens when friendly units are detected.



### 5.3.3.5 Ending the blocking positions

The occupation of a blocking position is started as a reaction to enemy behaviour, and should be ended once this enemy behaviour has seized. This decision could be made by either the ordered company agent or by the superior battalion agent. We have chosen to let the battalion agent make all the decisions in this matter, because this agent already has all the requisite background knowledge on which it based the initial order. The blocking position is ended once the blocked units are destroyed. As such, when the battalion agent observes that the equipment it ordered to be blocked has been destroyed, it will send a continue task to the ordered company agent. This structure allows room for future extra reasoning.

### 5.3.3.6 Diagrams

In Appendix C, you can find extended, more detailed, versions of the UML activity diagrams, Figures 5.3 and 5.4, accompanying this implementation description.

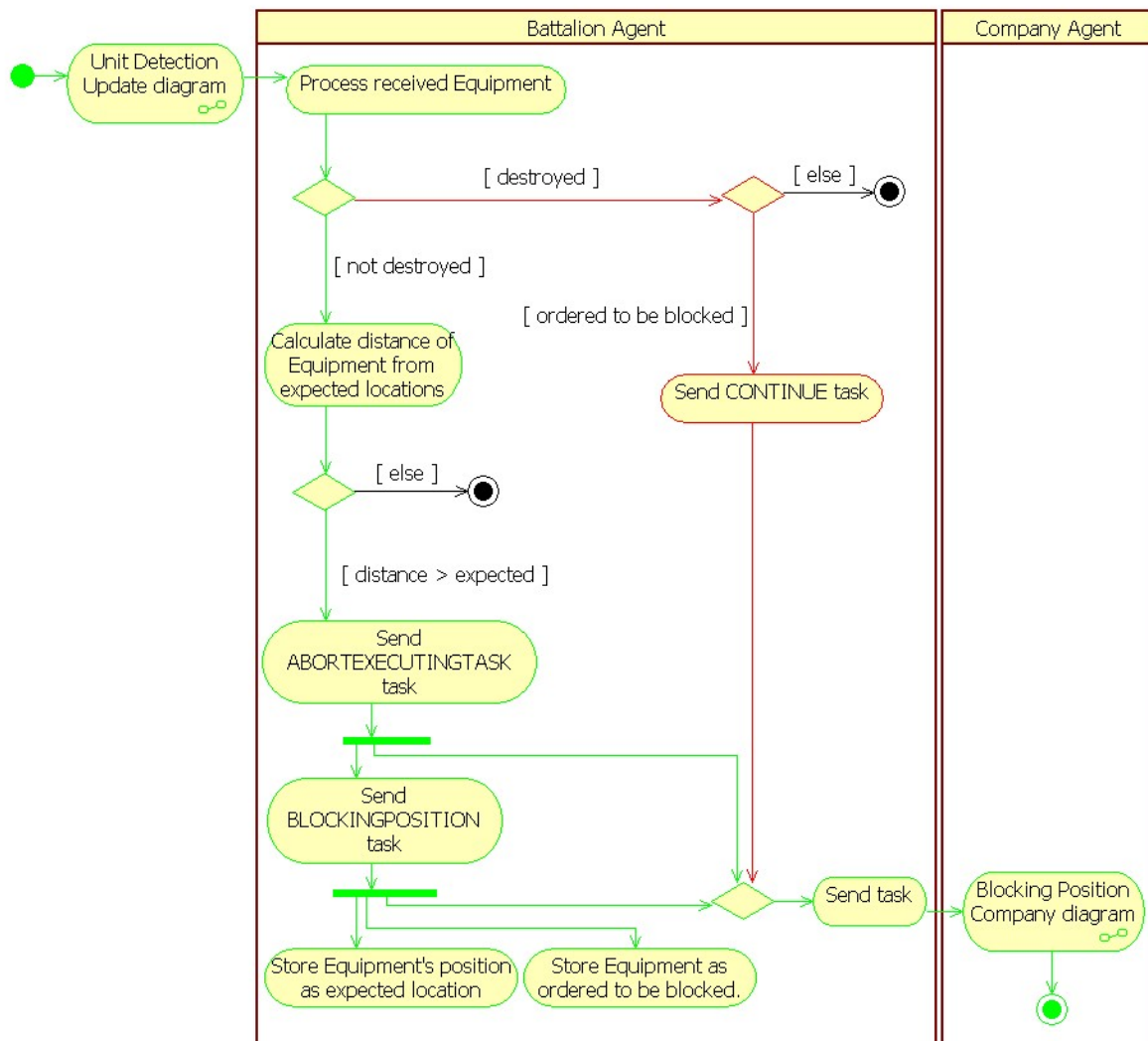
In Figures 5.3 and 5.4, we can see the discussed implementation as UML activity diagrams. Our blocking position is initiated in Figure 5.3, along the green path. A moving unit in the simulator causes a unit status update to be sent from the simulator to the HLA Manager. The HLA Manager then forwards this update in an agent-understandable way to the Information Manager. This Information Manager checks the allegiance of the received Equipment: friendly Equipments get stored with other friendly units and updated to their commanding agents, while enemy units get stored with other enemy units and the commanding agent gets updated. In our setup, the Battalion agent is the commanding agent.

In Figure 5.4, the green line shows us how the Battalion agent responds to the update from the Information Manager. First it is checked whether the received Equipment is still alive. If so, distances from the Equipment to expected locations are measured and compared to a predefined expected distance. If the Equipment is too far from expected locations, a blocking position is required and the Battalion agent initiates this by sending both the `ABORTEXECUTINGTASK` and `BLOCKINGPOSITION` tasks to a Company agent. Furthermore, data is stored to stop the Battalion agent from blocking this unit again at the next update. The Company agent then continues with the execution of these tasks.

In case the Equipment update received by the Battalion agent shows that an Equipment is destroyed, which was previously ordered to be blocked, the Battalion agent sends the `CONTINUE` task to the Company agent instead. This indicates that the blocking position, set up for the destroyed Equipment, was successful and that the agent can continue with previous tasks. This ending of the blocking position is indicated with the red lines in Figure 5.4.

## 5.3.4 CIS formations

The formations available in VR-Forces are stored in FRM configuration files, one for each formation. Furthermore, every type of aggregate can use a different subset of these formations, described in their OPE configuration file. The aggregate types we use are the platoon of M1A2 tanks and the company of M1A2 tanks. Therefore, the OPE configuration files linked to these aggregate types have been extended to allow our newly defined formations.



**Figure 5.4:** The blocking position reaction from the Battalion agent shown in an UML activity diagram. The green path shows the decisions that lead up to ordering the company agent to apply a Blocking Position. The red path shows how a CONTINUE task is sent to the company agent, ending a previous Blocking Position.

To move our platoons into formation, we have created the move into formation task in the FOM. Our platoons are ordered to move into formation at a location with a heading and a formation, after which they just forward this to the HLA Manager and, subsequently, the simulator. However, this does not work for our company agents, as we provide the simulator orders at the platoon aggregate level. For the company level formations, we have also created FRM files, but we cannot task our company aggregates. However, from the simulator's GUI,

the company aggregate can be tasked to move into formation using these FRM files.

To achieve these formations for our company agents as well, we need to task our platoon agents to move into formation at certain locations, and with certain headings, so that the company formation is formed. To this end, we have created extra Java classes to determine the correct locations and headings for each platoon, based on the FRM files as its input.

#### 5.3.4.1 Positional offset

Just as the move into formation task provides the simulator with positional offsets, in the form of the .FRM file, our company agent needs to provide these move into formation tasks with positional offsets. As such, we have created additional formation files (.FRM) for the company level, and the new Formation Java class capable of reading and storing these formation files and transforming input positions. Note that the company formation has to consider the platoon formation used, because distances between its platoons are measured from the edges of those platoon formations. As such, the positional offset in a company column formation will be different if the platoon is in a column formation compared to a platoon's line formation. For this thesis, we have assumed that the platoons occupy the same formation as their superior company, partly because we do not have the expert information needed to decide otherwise. Therefore, offsets could be and were calculated beforehand and stored in a specific FRM file.

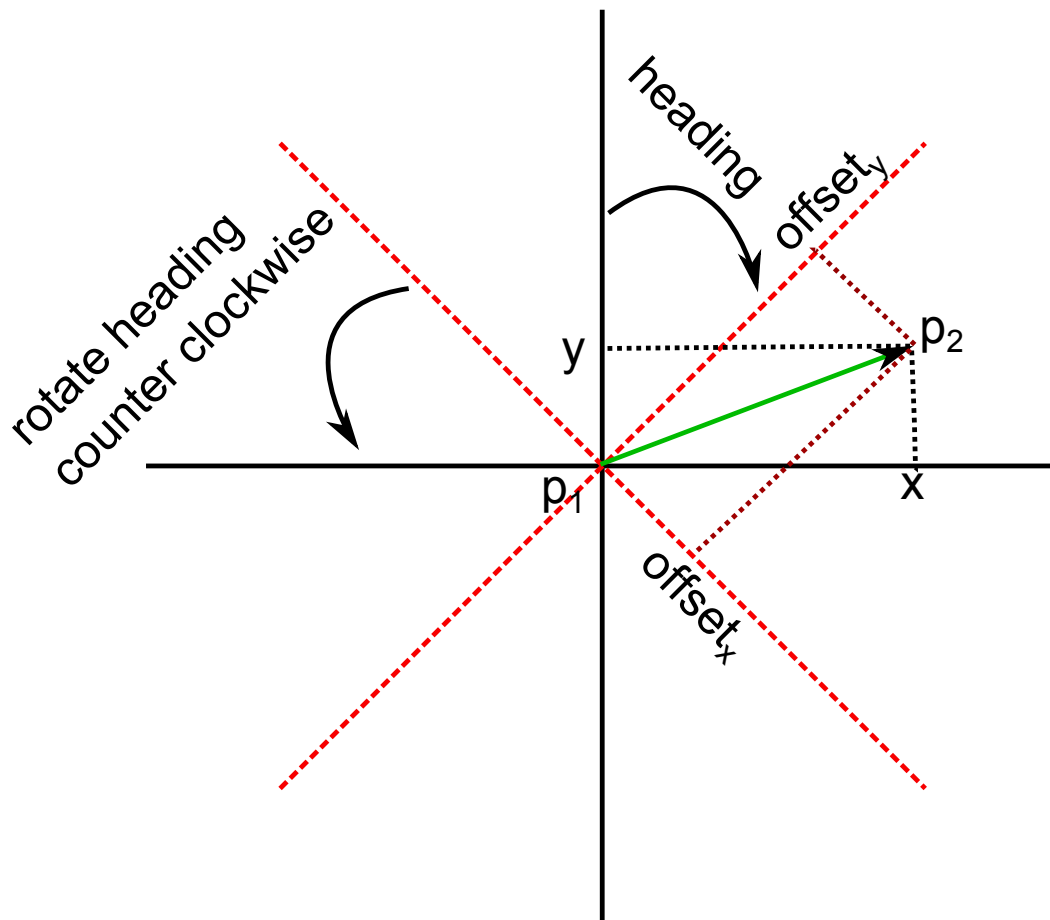
The Formation class provides the offset, compared to the starting location, for the next platoon in a HashMap, mapping from String to Double. The values in this HashMap are "right" for the offset (meters) to the right, "inFront" for the offset to the front and "above" for the offset in altitude. If the heading is towards the north,  $0^\circ$ , an offset can be directly applied to our UTM coordinates. Our offset "right" will extend the x-coordinate, "inFront" the y-coordinate, and "above" the z-coordinate. Therefore, we shall refer to the offsets as  $offset_x$

for "right",  $offset_y$  for "inFront" and  $offset_z$  for "above". So an offset of  $\begin{bmatrix} offset_x \\ offset_y \\ offset_z \end{bmatrix}$  on UTM

coordinates  $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$  will result in new UTM coordinates  $\begin{bmatrix} x + offset_x \\ y + offset_y \\ z + offset_z \end{bmatrix}$ . Note that negative values provide offsets to the left, behind and below and that the  $z$  coordinate for altitude is omitted in future references, since our agents only work with ground objects.

However, if the heading is not  $0^\circ$ , along the y-axis, the offset cannot be directly applied to our Cartesian UTM coordinates. Consider for example, that if the heading is set to  $90^\circ$ , the object's front would be along the x-axis and an offset to the right would be a negative offset on the y-axis. The actual offset location can now be calculated with a multitude of methods, among others: polar coordinates, vectors, corners and the Pythagorean Theorem or transforming coordinates using a rotation matrix. After comparing methods, we chose to use the rotation matrix in our agents. It computed the results faster than other tried methods.

**Rotation Matrix** An easy way to understand this method is to see the offset as part of its own coordinate system, similar to the Cartesian UTM coordinate system, but centred around the starting location  $P_1$  and tilted along the desired heading  $\theta$ . The offset-y-axis is for the meters



**Figure 5.5:** Example for the transformation matrix. Offset coordinate system shown in red, cartesian coordinate system in black. Coordinates  $offset_x$  and  $offset_y$  are known and can be translated to  $x$  and  $y$  by rotating the offset coordinate system counter clockwise with the heading  $\theta$ .

in front of the object, the offset-x-axis for the meters to the right of the object. When the heading is  $0^\circ$ , the two coordinate systems overlap and offset-coordinates  $\begin{bmatrix} offset_x \\ offset_y \end{bmatrix}$  can be directly translated to UTM coordinates  $\begin{bmatrix} x \\ y \end{bmatrix}$ . If not, the offset-coordinate system will have to be rotated through the counter clockwise heading  $\theta$ . This is accomplished by the matrix multiplication of the offset coordinates and the rotation matrix  $R'$ :

$$R' = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$$

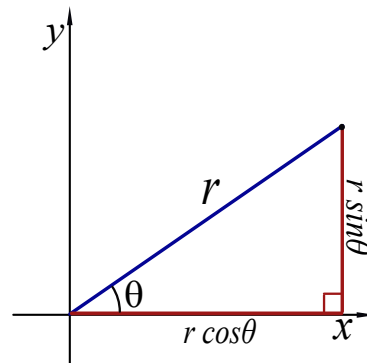
$$\overrightarrow{P_1 P_2} = \begin{bmatrix} x \\ y \end{bmatrix} = R' \begin{bmatrix} offset_x \\ offset_y \end{bmatrix}$$

$$= \begin{bmatrix} \text{offset}_x \cdot \cos \theta + \text{offset}_y \cdot \sin \theta \\ \text{offset}_x \cdot -\sin \theta + \text{offset}_y \cdot \cos \theta \end{bmatrix}$$

This vector  $\overrightarrow{P_1P_2}$ , the rotated offset, can then be added to position  $P_1$  to get position  $P_2$ .

$$P_2 = P_1 + \overrightarrow{P_1P_2}$$

#### 5.3.4.2 Heading



**Figure 5.6:** By converting from cartesian coordinates to polar coordinates, we can get our direct angle and heading  $\theta$ .

The heading of the formation is taken as the direction directly facing the target location (or entity). When we already know the current location and the target location, the heading (or angle / azimuth) to be turned to is found by translating from Cartesian coordinates to Polar coordinates and taking the azimuth  $\theta$ , or the angle of the vector from  $P_1$  to  $P_2$ .

$$\theta = \text{atan2}(y, x)$$

Here  $\text{atan2}^2$  is a common variation of the arctangent function, found in Java and other programming languages, that takes the quadrant into account.

#### 5.3.4.3 CIS: Column Formation (Tank Company)

The Column formation is a formation for the company level and the platoon level. The company Column formation, shown in Figure 4.3a, is stored in `CIScompanyColumnPlatoonColumn.frm`. From the simulator's GUI, the company aggregate can be tasked to move into the Column formation using this frm file.

#### 5.3.4.4 CIS: Column Formation (Tank Platoon)

The platoon Column formation, shown in Figure 4.3b, is stored in `CISplatoonColumn.frm`.

<sup>2</sup>Atan2: <http://en.wikipedia.org/wiki/Atan2>

### 5.3.4.5 CIS: Line Formation Traveling (Tank Company)

The Line formation is a formation for the company level and the platoon level. The company line formation, shown in Figure 4.4a, is stored in `CIScompanyLinePlatoonLine.frm`. From the simulator's GUI, the company aggregate can be tasked to move into the line formation using this frm file. The CIS Line Formation Traveling for the Tank Company prescribes a formation where, relative to leading platoon 1, platoon 2 is 100 meters to the left, platoon 3 is 100 meters to the right and other platoons start at 500 meters behind platoon 1, with a 100 meter (behind) interval between each of those platoons. Actually, Line Formation Traveling indicates positions for the Company commander, the Ltnt forward observer, the Executive Officer, the Sgt forward Observer and the Combat train, but these are omitted from our scenario, replaced by platoon positions.

Further platoons will be automatically positioned 100 meters extra behind the last platoon by VR-Forces (meaning we could have stopped the file at entry-4 instead).

### 5.3.4.6 CIS: Line Formation Traveling (Tank Platoon)

The platoon Line formation, shown in Figure 4.4b, is stored in `CISplatoonLine.frm`, shown in Code 5.10. Every entry is filled by a tank, with an offset from the leader's location (who has promotion-id 0).

**Code 5.10:** *Line formation for platoons in an FRM file*

```

1 (line-formation
2   (entry-0
3     (promotion-id 0)
4     (leader-promotion-id -1)
5     (position-offset 0.000000 0.000000 0.000000)
6   )
7   (entry-1
8     (promotion-id 1)
9     (leader-promotion-id 0)
10    (position-offset -30.000000 -50.000000 0.000000)
11  )
12  (entry-2
13    (promotion-id 2)
14    (leader-promotion-id 0)
15    (position-offset -30.000000 -150.000000 0.000000)
16  )
17  (entry-3
18    (promotion-id 3)
19    (leader-promotion-id 0)
20    (position-offset 0.000000 -100.000000 0.000000)
21  )
22  ...
23 )

```

## 5.4 DSS

In this implementation section about the decision support system we will describe how this system has been implemented. This starts with the creation of the GUI, a Java table, in Section 5.4.1, followed by explanation of the computation of the MOEs that will be displayed in the DSS in Section 5.4.2.

### 5.4.1 Implementing the GUI

As stated in Section 4.3, the Information Manager agent is the connection from our system architecture to the new DSS. At startup, it initiates the DSS creation, as shown in Figure 5.1, and further processing of the MOEs and adjusting the DSS's GUI is also controlled by the Information Manager agent.

The DSS GUI has been made as a Java table, according to the DSS model, more specifically a Java class DSS which extends JPanel, a window component of Java's Swing which can be used to create graphical user interfaces (or GUI). Inside this JPanel we can place components, which shall be our table.

#### 5.4.1.1 Table

The table that we place in our DSS is a JTable (Java table from the Swing set), initialized with a customized MyTableModel. In this model, a table is created with as many columns as there are column names provided to it. Thus, we provide the name of each MOE we want to this table model, after which the JTable in our JPanel will be able to show these columns. More importantly, we also provide the table with a column model that will allow us to hide the columns in the table. By hiding appropriate columns, the user can select only those criteria for comparison that are desired. The column model used is the XTableColumnModel from Stephen Kelvin, available at <http://www.stephenkelvin.de/XTableColumnModel/>.

After the table has been created, it is added to our JPanel together with the actions to add a row, to show or hide all columns and to retrieve the results from the agent. For more information on creating or adjusting such a table GUI in Java, see Oracle's tutorials at <http://docs.oracle.com/javase/tutorial/uiswing/components/table.html>.

#### 5.4.1.2 Listening to the DSS

The agent has a plan *ProcessDSSGUI* that implements Java's TableModelListener and ActionListener. This provides the plan with the tableChanged() and actionPerformed() methods.

**TableModelListener** The tableChanged() will trigger whenever a user adjusts our DSS table, i.e. by adding a new row. In Code 5.11 we can see the small piece of Java code that is executed at that point. All it actually does is set beliefs *newCOA* and *rowCOA* to true and the row that has been added, causing the next order to be matched to this new row, plus it outputs some warnings to the user. If newCOA is still true by the time this new row is added, it means that no order has been matched to the previous row(s) yet. This might be noteworthy.

**Code 5.11:** Java code for the `tableChanged()` method in the Information Manager's `ProcessDSSGUI` plan

```

1 case TableModelEvent.INSERT:
2 //The inserted rows are in the range [firstRow , lastRow]
3 for(int r=firstRow; r<=lastRow; r++){
4 // Row r was inserted
5 if((Boolean)bb.getBelief(Battalion.newCOA).getFact() == true){
6 String warn = new String(" Note: One or more previously added rows did not
7 have a COA linked yet. \n The last added will instead link to the
8 next COA. ");
9 mLogger.warn(warn);
10 JOptionPane.showMessageDialog(decisionSupportSystem.table ,
11 warn,
12 " Overwriting previous row" ,
13 JOptionPane.WARNING_MESSAGE);
14 }
15 bb.getBelief(Battalion.newCOA).setFact(true);
16 bb.getBelief(Battalion.rowCOA).setFact(r);
17 }
18 break;

```

**ActionListener** The `actionPerformed()` method will trigger whenever the user presses one of the buttons on the DSS, such as the one to retrieve the results. This is currently the only action that's possible to be performed which triggers this action listener. When pressed, the Information Manager will retrieve results for every column from its beliefs.

## 5.4.2 Computing the MOEs

As discussed, the computation of the MOEs is triggered by the action listener after the user presses a button on the DSS. The following columns (MOEs) have been added and shall be discussed in this section:

- Survival rate
- Fuel percentage
- Ammunition percentage
- Completed
- Enemy survival rate

### 5.4.2.1 Survival rate

The survival rate measure of effectiveness is the ratio of friendly units that are still alive compared to the number of units the mission started with. In Code 5.12 we show the Java code used to retrieve the MOE and put it in our DSS. This section of code will be triggered once per press of the button. The agent retrieves all friendly battalions from its beliefs (`friendlyForces` is an already-retrieved belief), after which it can just ask each battalion `Aggregate` for its strength,



which is continuously updated as the simulator provides new information to the agent. The strength of a battalion has been taken as the mean strength of its subordinate companies. Their strength is the mean strength of their subordinate platoons, who's strength is the number of subordinate equipment that are still active divided by the total number of equipment. So in effect, the battalion's strength is the percentage of tanks that are still active in its battalion.

**Code 5.12:** Java code for retrieval of the survival rate MOE in the DSS

```

1      if (columnName.equals(Battalion.columnSurvival)) {
2          columnSurvival = i;
3          Vector<Aggregate> battalions = friendlyForces.getAllBattalions();
4
5          for (Aggregate battalion : battalions) {
6              battalionStrength = battalion.getStrength();
7              String battalionStrengthPerc =
8                  NumberFormat.getPercentInstance().format(battalionStrength/100.0);
9              model.setValueAt(battalionStrengthPerc, row, columnSurvival);
10             break;
11         }
    }

```

#### 5.4.2.2 Fuel percentage & ammunition percentage

The fuel percentage is, like the survival rate, a mean percentage of an attribute of every Equipment / tank in the battalion. In fact, it is retrieved in the same way as the survival rate, except this time by calling *battalion.getResourceAmount("fuel")*. The fuel percentages are updated to our Equipment objects from the event updates we receive since we added the entityFuel event to the FOM, instead of directly from the tank object we receive from VR-Forces. The ammunition percentage has not been implemented on the agent-side during this project. Unlike fuel, ammunition has multiple types per unit, as a soldier might carry a machine gun and a pistol and some grenades. It is less clear if these should then be represented as one percentage (the mean over all possible ammunition), or if the MOE should be spread out over multiple sub-columns with percentages for each ammunition type.

#### 5.4.2.3 Completed

Completed is a basic criterium of a mission: did the COA actually come to completion or was it stopped by the enemy resistance? In Code 5.13 we can see the code used to retrieve this criterium. First the agent retrieves from its beliefs the set of status reports it has gotten from its company agents and then the set of orders that it has been tasked by the C2 system. These are matched to see if the order from the C2 system has a completed task report. If so, the criteria is set to true, else it will be false. There has been no check implemented to see if the order belongs to the current COA (row), so this only works in our current setup where the agents can actually only perform one COA at a time.

**Code 5.13:** Java code for the retrieval of the completed criteria in the DSS

```

1  if (columnName.equals ( Battalion . columnCompleted )) {
2      columnCompleted = i ;
3      boolean completed = false ;
4      OrderStatus [] orderStati = ( OrderStatus [] )
          bb . getBeliefSet ( Battalion . ordersCompleted ) . getFacts () ;
5      Order [] orders = ( Order [] ) bb . getBeliefSet ( Battalion . orders ) . getFacts () ;
6
7      for ( OrderStatus orderStatus : orderStati ) {
8          for ( Order order : orders ) {
9              if ( ( order . getOrderId () . equals ( orderStatus . getOrderId () ) ) &&
10                 ( orderStatus . getStatus () . getName () . equals ( Battalion . complete ) ) )
11                  {
12                      completed = true ;
13                  }
14              }
15          }
16      model . setValueAt ( new Boolean ( completed ) , row , columnCompleted ) ;
17  }

```

#### 5.4.2.4 Enemy survival rate

Finally, the enemy survival rate is retrieved by taking the enemyForces belief (the set of enemy Equipment) from the agent's beliefs, and then checking the number of Equipment that are destroyed in this list. Then the number of non-destroyed Equipment divided by the total number of Equipment provides the enemy's strength and is set to as the column value. See Code 5.14 for this method. This set of enemy Equipment is updated whenever an Equipment is updated from VR-Forces.

**Code 5.14:** Java code for the retrieval of the enemy survival rate for our DSS

```

1  if (columnName.equals ( Battalion . columnEnemy )) {
2      columnEnemy = i ;
3      Vector < Equipment > enemyForces = ( Vector < Equipment > )
          bb . getBelief ( Battalion . enemyForces ) . getFact () ;
4      float destroyed = 0 ;
5      float total = ( float ) enemyForces . size () ;
6
7      for ( Equipment enemy : enemyForces ) {
8          if ( enemy . getState () . equals ( Battalion . stateDestroyed ) ) {
9              destroyed ++ ;
10         }
11     }
12     float alive = total - destroyed ;
13     float enemyStrength = alive / total ;
14     String enemyStrengthPerc =
          NumberFormat . getPercentInstance () . format ( enemyStrength ) ;
15     model . setValueAt ( enemyStrengthPerc , row , columnEnemy ) ;
16 }

```

## Chapter 6

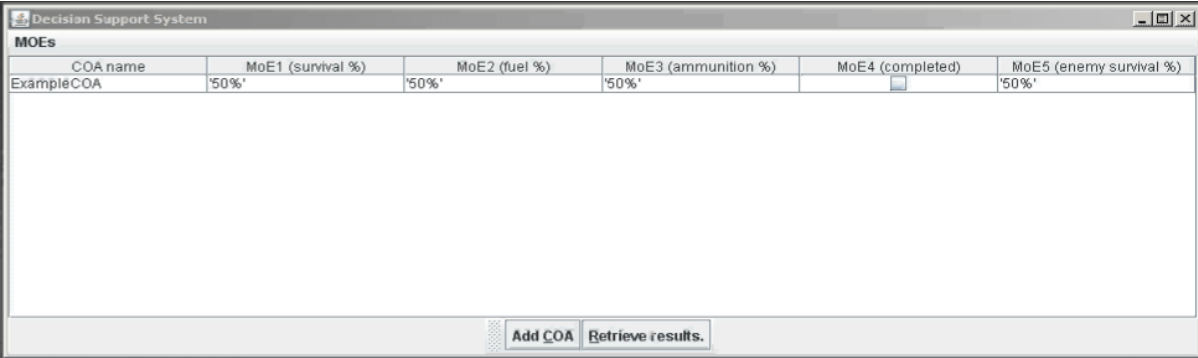
# Results

In this chapter we shall show how our systems turned out and what the new capabilities are. As told before, our implementation concerns two systems: the multi-agent system and the decision support system. Therefore, this chapter shall consist of two sections, one for each of these systems.

Our results in this document will be supported by screenshots from the simulator during execution of the CISs and monitoring of the DSS. Furthermore, for video demos you can visit YouTube channel *TorecLuikProjecten* at <http://www.youtube.com/user/TorecLuikProjecten>. There are seven demos in total as shown below, the first three of the list were used in the presentation of this project.

- *Demo of Command Agents - Assault*, showcases the execution of the Assault.
- *Demo of Command Agents - Blocking Position*, showcasing the execution of Blocking Positions.
- *Demo of Command Agents - Decision Support System (DSS)*, showcasing the DSS as it records data during the Assault.
- *Demo of Command Agents - Decision Support System (2)*, which shows similar behaviour as Demo of Command Agents - Decision Support System (DSS), but now with increased visual quality.
- *Demo of Command Agents - Decision Support System (Multiple tables)*, showcasing how a second DSS table can be used for comparison.
- *Demo of Command Agents - Attack (old)*, in which the old Attack order is shown.
- *Demo of Command Agents - Attack (old, random start setup)*, which also shows the Attack, yet now with different positions, emphasizing that no formations are held during the Attack.

## 6.1 DSS Behaviour



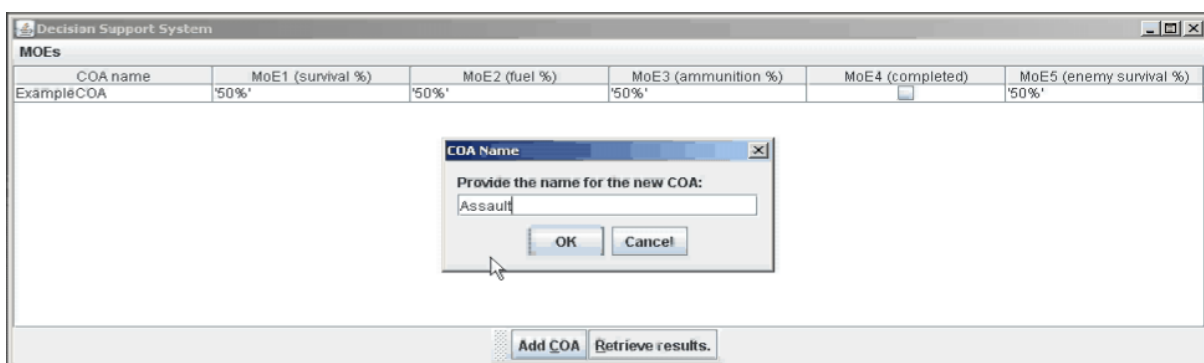
| COA name   | MoE1 (survival %) | MoE2 (fuel %) | MoE3 (ammunition %) | MoE4 (completed)         | MoE5 (enemy survival %) |
|------------|-------------------|---------------|---------------------|--------------------------|-------------------------|
| ExampleCOA | '50%'             | '50%'         | '50%'               | <input type="checkbox"/> | '50%'                   |

**Figure 6.1:** *The Decision Support System on startup.*

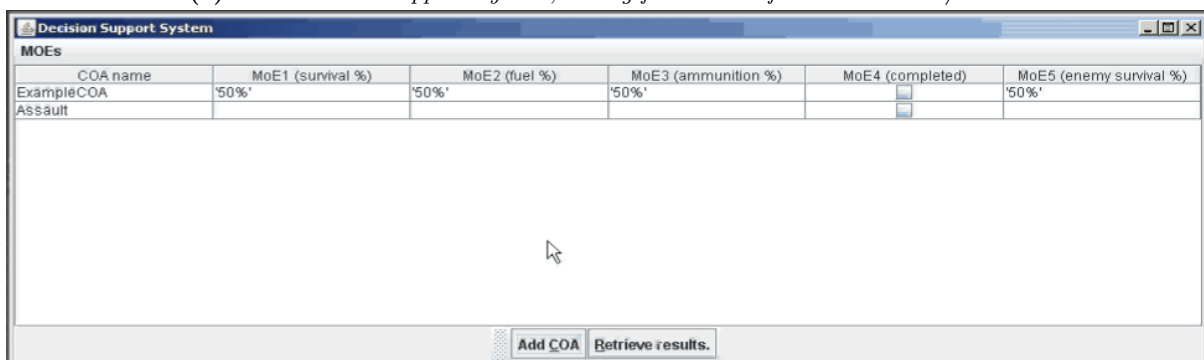
Our decision support system has been implemented based on the initial concept from Section 4.4. As such, we have created a table that is launched by the Information Manager on startup, after launching all other agents. See Figure 6.1 for a view of the DSS when initialized.

### 6.1.1 Connecting DSS with an order

Rows can be added with the *add row* button, after which the name of the new COA row (first column) has to be provided. If a duplicate name is provided, the system will notify the user and automatically add an enumerator to the name. See Figure 6.2 for the DSS with a row added. Whenever a row is added to the table, it is automatically connected to all the next orders. This means that the MOEs, generated for the row's columns, will be based on the events happening afterwards.



(a) The Decision Support System, asking for a name for the new row / COA.

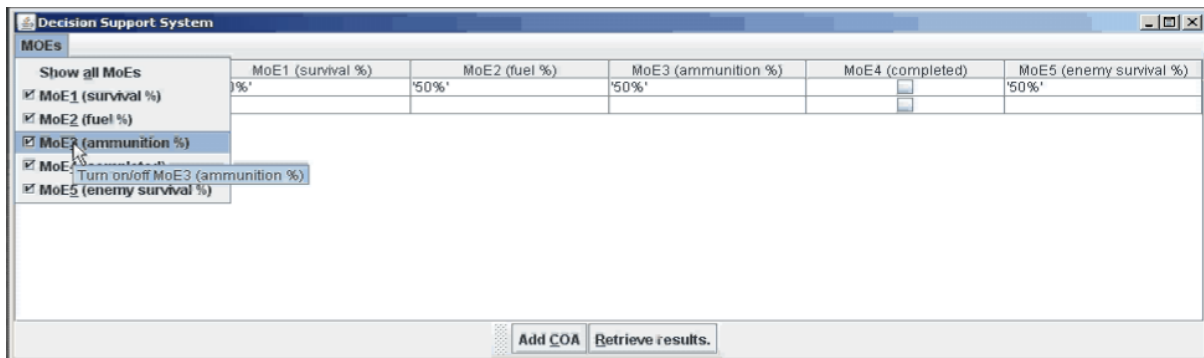


(b) The Decision Support System with the new row added.

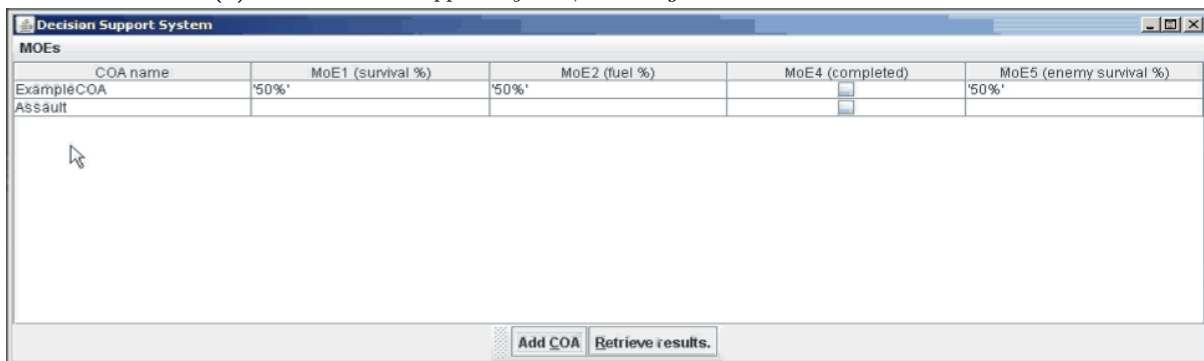
**Figure 6.2:** The Decision Support System as we add a new row.

### 6.1.2 Hiding MOEs

The table is initialized with columns for all measures of effectiveness, and one example row. After this point, the columns can be hidden (if there is no interest in a particular MOE) and later revealed again. See Figure 6.3 for the same DSS as in Figure 6.1 but with a column hidden.



(a) The Decision Support System, selecting what columns we want to see.



(b) The Decision Support System after we have hidden a column (fuel rate)

**Figure 6.3:** The Decision Support System as we hide a column (fuel rate).

### 6.1.3 Retrieving MOEs for the order

The data for in the columns is retrieved and/or calculated from the Information Manager's beliefs, when the button *Retrieve results* is pressed. The new data then overwrites the old data in the newest row. As such, any type of MOE can be updated in this way. During this project, we can only retrieve up to date results concerning survival rates (%), fuel rate (%) and completed. See Figure 6.4 for the DSS with retrieved results.

| COA name   | MoE1 (survival %) | MoE5 (enemy survival %) | MoE2 (fuel %) | MoE4 (completed)         |
|------------|-------------------|-------------------------|---------------|--------------------------|
| ExampleCOA | '50%'             | '50%'                   | '50%'         | <input type="checkbox"/> |
| Assault    | 100%              | 100%                    | 100%          | <input type="checkbox"/> |

(a) The Decision Support System, after results are retrieved at the start of the scenario.

| COA name   | MoE1 (survival %) | MoE5 (enemy survival %) | MoE2 (fuel %) | MoE4 (completed)         |
|------------|-------------------|-------------------------|---------------|--------------------------|
| ExampleCOA | '50%'             | '50%'                   | '50%'         | <input type="checkbox"/> |
| Assault    | 88%               | 35%                     | 100%          | <input type="checkbox"/> |

(b) The Decision Support System, after results are retrieved near the end of the scenario.

**Figure 6.4:** The Decision Support System after results are retrieved.

#### 6.1.4 Comparing MOEs

During this project we did not yet try to connect one DSS to multiple simulation runs, which should be possible after some adjustments, in theory. Instead, comparing MOEs can be done in one simulation between different time points, by adding a new row. Also, different COAs could be compared by starting a new set of agents and a new simulator. This way, you would get two DSS windows showing results from two different simulator runs. See Figure 6.5 for an example of two DSS screens from two different simulations.

The figure consists of two screenshots of a software interface titled "Decision Support System".

The top screenshot shows a table with the following data:

| COA name   | MoE1 (survival %) | MoE5 (enemy survival %) |
|------------|-------------------|-------------------------|
| Assault    | 89%               | 35%                     |
| ExampleCOA | 50%               | 50%                     |

The bottom screenshot shows a table with the following data:

| COA name   | MoE1 (survival %) | MoE5 (enemy survival %) | MoE4 (completed)                    |
|------------|-------------------|-------------------------|-------------------------------------|
| ExampleCOA | 50%               | 50%                     | <input type="checkbox"/>            |
| Assault 2  | 72%               | 75%                     | <input checked="" type="checkbox"/> |

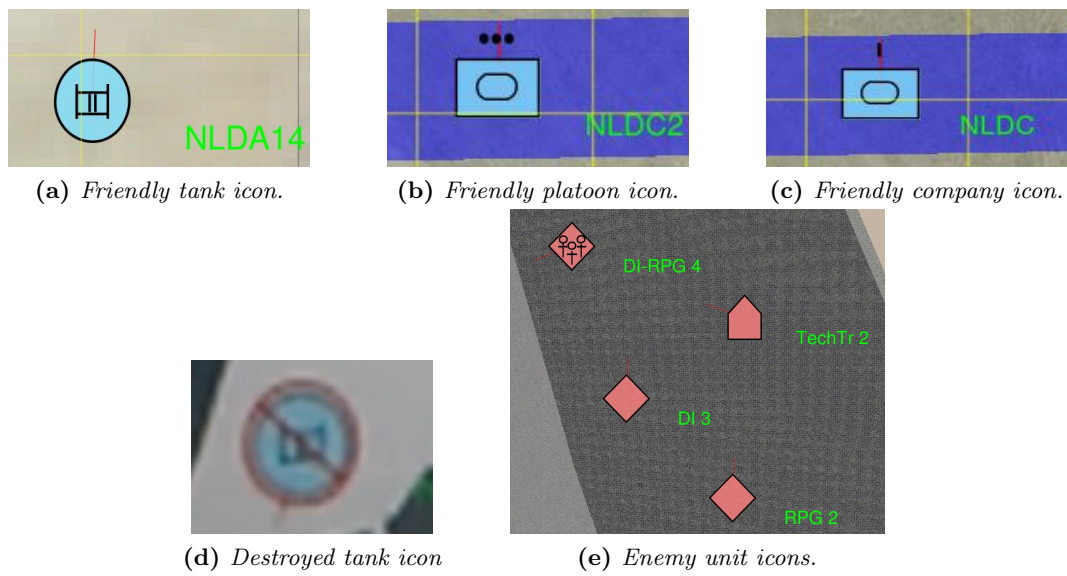
**Figure 6.5:** *The Decision Support Systems after results are retrieved in two separate simulations, with two disparate COAs.*

## 6.2 CGF Behaviour

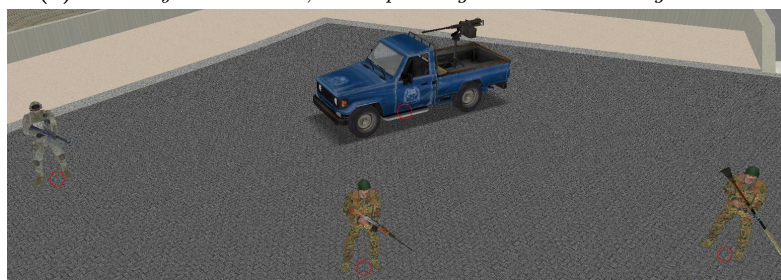
We have implemented CISs that describe CGF behaviour in our agents. In this section, we will show what behaviour in the simulator is the result of the implementation of these CISs. For each CIS, we will show screenshots from a video and describe the shown behaviour according to our implementation. Furthermore, we shall contrast these with similar behaviour, if that was available from the previous project.

**Legend** To better understand what is going on in the video and its stills, see our starting positions in Figure 6.8. In the bottom left we see the friendly forces in blue and in the top right we see the enemy forces in red. The figures from the simulator, as seen in the video, are standard figures used in NATO militaries and thus used by VR-Forces. For more information, see Department of Defense (2005). In Figure 6.6 we show the icons as used in the 2D view of the simulator and in Figure 6.7 we show the 3D versions of the units we are simulating.





**Figure 6.6:** The icons for different units used in our scenario.



**Figure 6.7:** The 3D versions of the units used in our scenario.

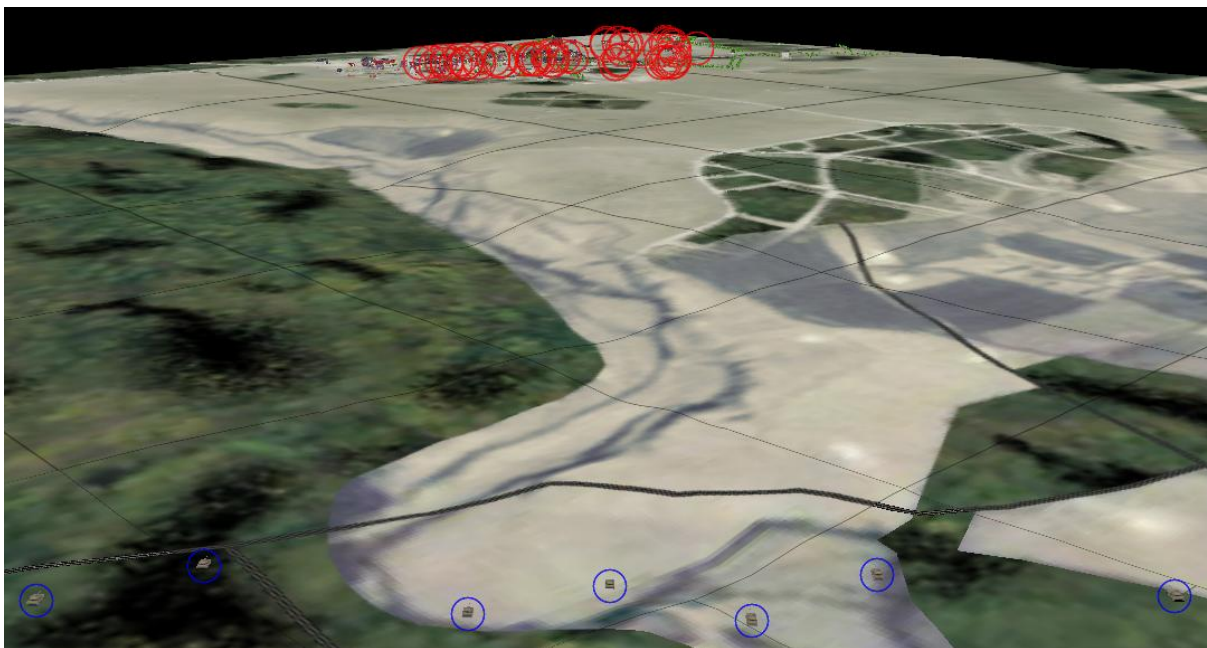


**Figure 6.8:** Friendly companies' starting positions at the south west corner, south east corner and north west corner. Enemy positions in the north east town. Blue fields at the aggregate icons indicate the area covered by its subordinate tanks.





**Figure 6.9:** *The objective from the enemy's perspective.*



**Figure 6.10:** *The objective from company B's perspective.*

## 6.2.1 Marching

We have implemented several new methods of transportation, generally referred to as marching. Previously, only one method of marching was possible, the Move. Our implementations provide the Tactical Road March, the Column formation and the Line formation methods of marching. In effect, these are formations that the platoon or company moves in, whereas the previous Move did not use any formation. All four methods shall be shown next.

### 6.2.1.1 Previous Behaviour - Move

In Bronkers (2011), the previous version of this MAS, there was one way of movement, using VR-Forces' B-HAVE. This Move is shown in Figure 6.11 as a comparison to the doctrinal march behaviour implemented during this project.



(a) A starting position for the Move.



(b) The Move after a short while, tanks have individually moved to the north east in a relatively straight line since no path planning is being used.

**Figure 6.11:** *The Move command from Bronkers (2011).*

### 6.2.1.2 Tactical Road March

One of our new implementations from a CIS was the Tactical Road March, see Chapter 4.1.2.3. The Tactical Road March implementation is shown in Figure 6.12a.

### 6.2.1.3 Column Formation

One of our new implementations from a CIS was the Column formation, see Chapter 4.1.2.3. The Column formation implementation is shown in Figure 6.12b.



(a) The Tactical Road March CIS in action.

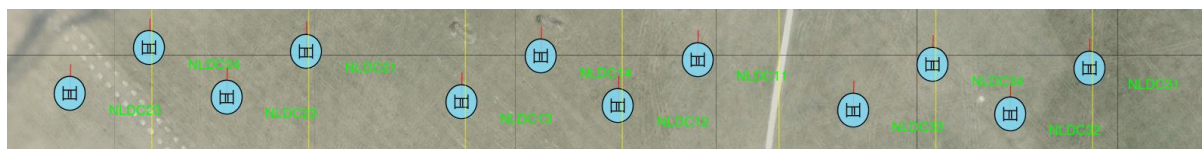
(b) The Column Formation CIS in action.

**Figure 6.12:** The Tactical Road March CIS and Column Formation CIS in action. Note that the Column formation requires a lot more space, so the scales are not equal.



### 6.2.1.4 Line Formation

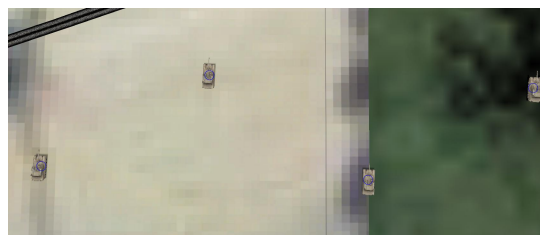
One of our new implementations from a CIS was the Line formation, see Chapter 4.1.2.3. The Line formation implementation is shown in Figure 6.13 and Figure 6.14.



**Figure 6.13:** *The Line Formation CIS in action.*



**(a)** *The Line Formation CIS on the platoon level in action, with icons.*



**(b)** *The Line Formation CIS on the platoon level in action, in 3D.*

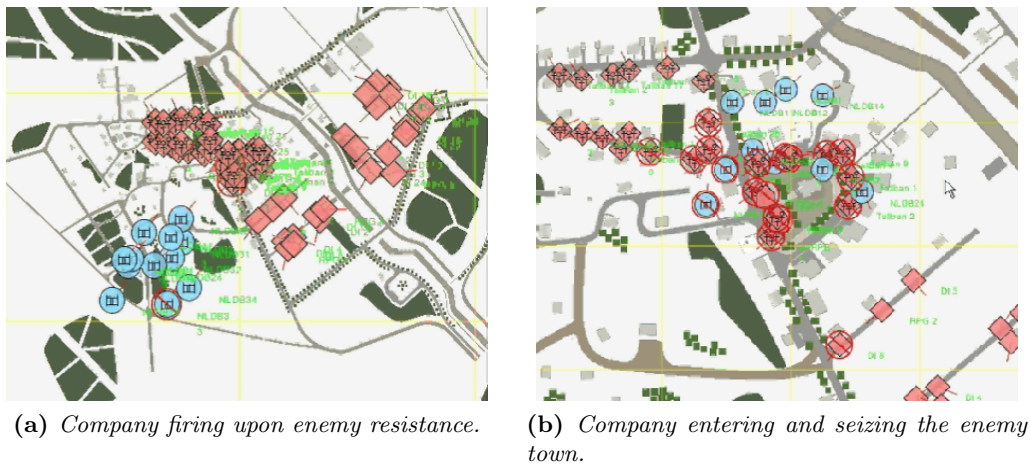
**Figure 6.14:** *The Line Formation CIS in action on the platoon level, with icons and in 3D.*

## 6.2.2 Assault on a enemy position

The Assault is the doctrine that contains the attack on enemy position, together with the prior and post behaviour. As such, it can be seen as the extension or improvement of the previous Attack or Seize behaviour, where Seize includes a defensive position after the Attack. For this scenario, our company of 3 M1A2 tank platoons shall assault a fortified enemy position in a town to the north east, with their starting position shown in Figure 6.8. See also the demo video *Demo of Command Agents - Assault* on YouTube, from which some of the following images are taken.

### 6.2.2.1 Previous Behaviour - Seize

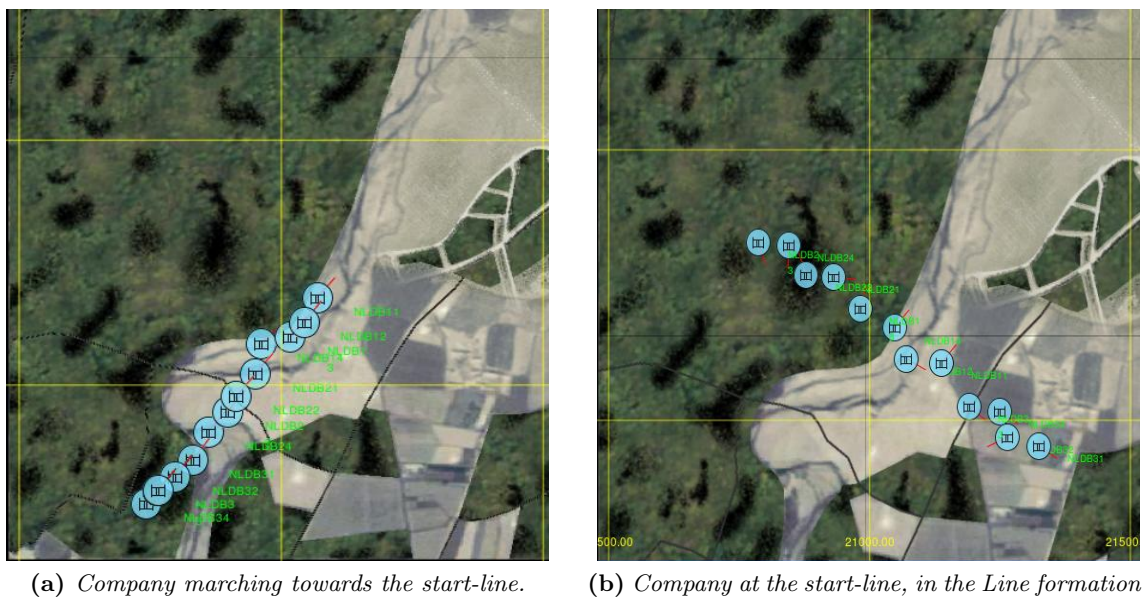
Previously implemented behaviour that is similar to our new Assault is the Seize behaviour. The Seize consists of two phases: first an Attack is executed on the target position, following by a Defend on that same position. The Attack is made up of a Move and the setting of ROE to fire at Will, while the Defend is made up of a Move and the setting of ROE to fire when fired upon. In Figure 6.11, we can see the company moving towards the enemy town for their attack. In Figure 6.15a, fire is exchanged between the friendly and enemy troops. Then in Figure 6.15b, the company has entered the town and finally the company takes up a defensive position in the town, which doesn't actually show any difference to Figure 6.15b, as only the ROE changes to *fire when fired upon*.



**Figure 6.15:** *The Seize in action.*

#### 6.2.2.2 Phase 1: Preparation

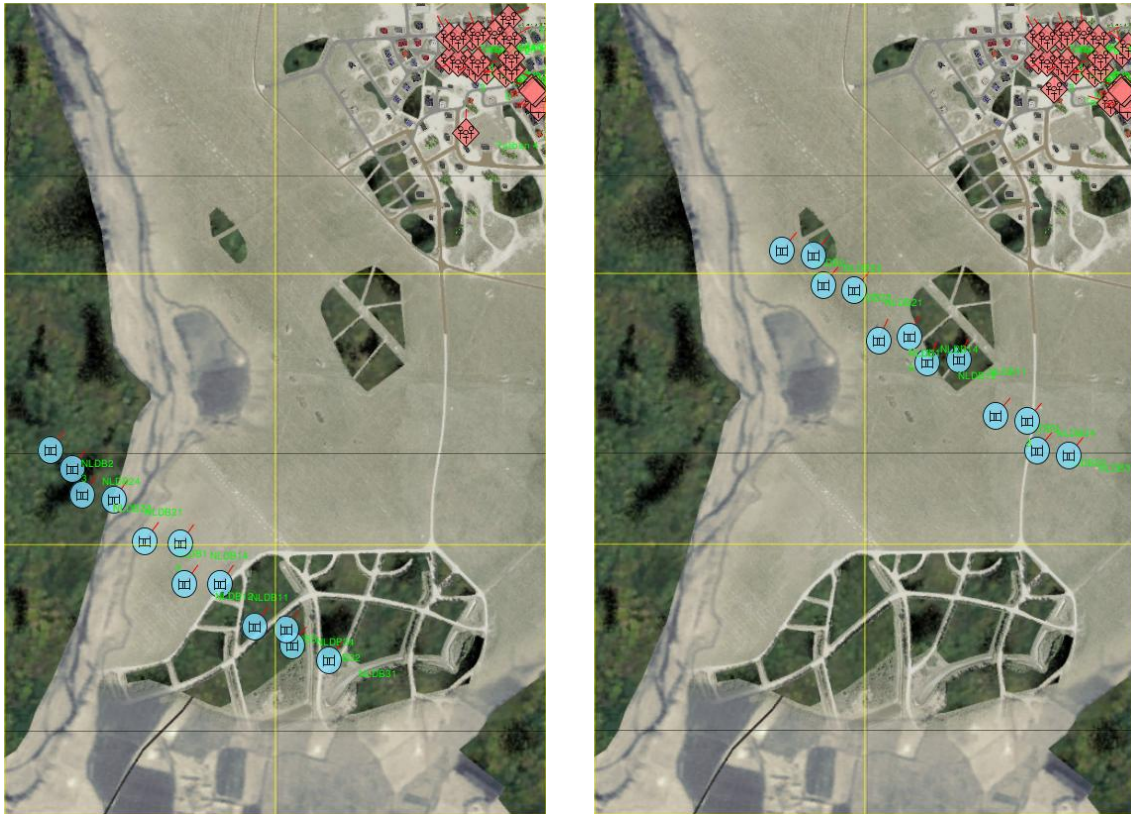
The assault doctrine contains a lot more steps than shown in the previous section. The first of all is the preparation: moving towards the start-line. The preparation starts with moving into their march formation as in Section 6.2.1.2. From this position, they march, as shown in Figure 6.16a, to the start-line. This start-line has been determined by the company agent, based on the 1500 meter distance from the final location. When they arrive, the company and its platoons move into the Line formation, see Figure 6.16b, concluding Phase 1.



**Figure 6.16:** *The Preparation phase of the Assault in action.*

### 6.2.2.3 Phase 2: Fire and Movement

Phase 2 consists of the movement between the start-line and the final assault line. When not under fire, the company will march in Line formation from the start-line to the final assault line, 600 meters from the target. In Figure 6.17a we can see the company during this movement. In Figure 6.17b, the company has reached its final position at 600 meters from the target.



(a) Company marching towards the 600 meters line.

(b) Company in Line formation at the 600 meters line.

**Figure 6.17:** *The 2nd phase of the Assault in action.*

### 6.2.2.4 Phase 3: Final Attack

Once the company has reached the last line, a final attack on the target position will take place. This means that its rules of engagement are set to fire at will as the company moves towards its target position. We can see how fire is opened in Figure 6.18a, how the company moves into the enemy's defensive position in Figure 6.18b and how the attack is concluded when the tanks arrive at the center of the village and most of the enemies (and friendly tanks) are destroyed in Figure 6.18c.

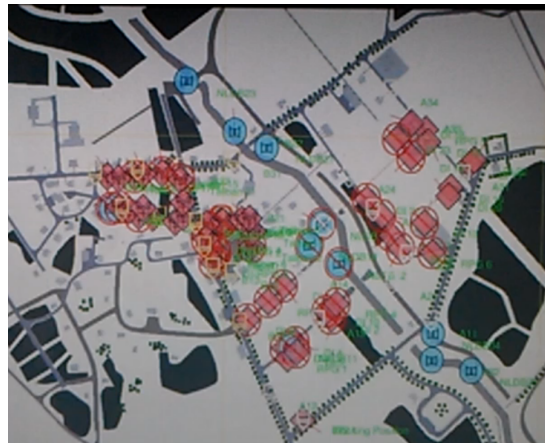




(a) Company opening fire on enemy targets.



(b) Company moving towards center of the town.

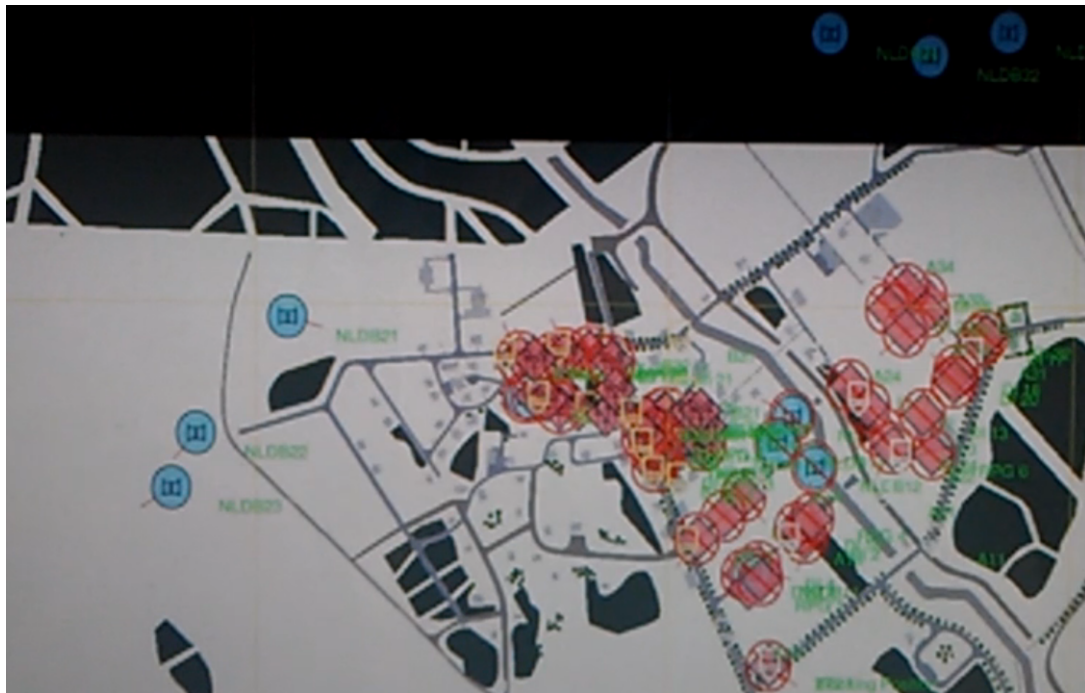


(c) Company completed the attack on the town.

**Figure 6.18:** *The 3rd phase of the Assault in action.*

#### 6.2.2.5 Phase 4: Consolidation

When the attack has been completed, the company has to consolidate and reorganize in a position beyond the target destination. This way, they can get ready for their next task. In Figure 6.19 we can see the company consolidating and reorganizing at the edge of the town, finalizing the assault. As can be seen, there are only 6 tanks left, divided over 2 platoons. One of the platoons is positioned in the blackness beyond the city, where the ground type has not yet been declared.



**Figure 6.19:** *Company finalized assault by consolidation at the edge of the town.*

### 6.2.3 Blocking Position

The (Hasty Occupation of a) Blocking Position CIS provides the agent with reactive behaviour, based on enemy presence at unexpected locations. In this section we shall show the different parts of the blocking position.

In these results we have a scenario in which an Assault is ordered, but later interrupted for a Blocking Position. See also demo video *Demo of Command Agents - Blocking Position* on the YouTube channel, from which most of these Figures are taken. First, as in Figures 6.16a, 6.16b and 6.17a, we can see the company marching to and from the start-line.

#### 6.2.3.1 Enemy Presence

Next, a (group of) enemy unit(s) is detected at a location with large distance from the expected location (where the units will Assault), see Figure 6.20. If the calculated distance is too great, a blocking position will be ordered, to keep these new enemies at bay.



(a) Unexpected enemy presence, closest to assaulting company B.

(b) Unexpected enemy presence, closest to company A.

**Figure 6.20:** Enemy presence at a great distance from the expected enemy location.

### 6.2.3.2 Blocking Position occupied

As ordered, the company agent initiates the Blocking Position formation at its current location and engages the enemy presence with fire at will. See Figure 6.21 for the blocking position of company A. Assaulting company B does not occupy the Blocking Position formation because the enemy is already destroyed before this stage would come.



**Figure 6.21:** Blocking positions occupied towards enemy presence and firing at will initiated.

### 6.2.3.3 Enemy destroyed

After some period of time, the unexpected enemy will be destroyed and the blocking position successful. See Figure 6.22.



(a) Enemy presence removed by the blocking position from company A.



(b) Enemy presence removed by the blocking position from company B.

**Figure 6.22:** Enemy presence removed by the blocking position.

#### 6.2.3.4 Blocking Position completed

At this point, the Blocking Position is considered successful and the company is allowed to continue on its march. See Figure 6.23, where company B continues with the next phase of the Assault.



**Figure 6.23:** *Company continuing towards objective.*

## Chapter 7

## Discussion



In this chapter we discuss our findings and how they relate to our objectives and those of the field. Also, we discuss the limitations and implications of our methods and findings and what further research could be conducted to solve some of these limitations.

As a reminder, our research question was:

”How can we create a useful decision support system for military planning, using the combination of simulation and C2?”.

To create a useful decision support system (DSS) using simulation, the system needs to be usable in the C2 process and the results provided need to be realistic before it can provide any support. Thus we can divide our answer into two parts that will both need to be solved:

- Usability
- Realism

As such, we shall show how our results relate to these categories in Section 7.1 and 7.2. Together, these two categories define the usefulness of the DSS application and answer our research question.

## 7.1 Usability

The usability issue mostly concerns the decision support system’s interface: if the system’s interface is not intuitive, it will hardly support the user. Luckily, other researchers have already delved into this subject, allowing us to provide requirements for DSS usability in C2 in Section 2.3.2. Furthermore, we have discussed the possibilities of a DSS with military experts from the RNLA, as discussed in Section 3.1. The results we can see in Section 6.1 relate to this usability problem of our original research question. It shows how we created a usable prototype of our decision support system. In the following subsections we shall describe our findings based on the requirements from the experts and literature.

### 7.1.1 Expert desires

In Figure 4.8 we captured the desire of the experts for a DSS in C2. In Section 6.1 our version is shown. How do these compare?

**Table** Multiple courses of action (COAs) would be provided in the C2 system, to be compared in the DSS table based on chosen measures of effectiveness (MOEs). True to form, our DSS provides a table in which COAs are represented in the rows and MOEs in the columns, as can be seen in Figure 6.1.

**Adding MOEs & COAs** The initial concept (Figure 4.8) also shows options for adding new MOEs, from a list, to the columns and new COAs to the rows. In Figures 6.3 and 6.2 we can see how our DSS allows hiding of MOEs (and inversely the adding of MOEs) and adding of new COA rows.

### 7.1.2 Literature requirements

Other researchers reported requirements for decision support systems to which we can compare our results, see Section 2.3.2. We shall discuss these requirements here, separated in a paragraph for every requirement.

**Standards** One subset of usability is also interoperability between simulation and C2 with the use of standards, so that the simulation (and DSS) can communicate with other applications (Hazen, 2011). We have delivered on that requirement with the integration of the 3 main standards in our simulation: C-BML for our orders, MSDL for the definition of the scenario for C2, agent and simulation, and HLA to allow distributed systems to connect and communicate using the other standards. Prelipcean et al. (2010) also highlight this need for integration with other information, planning and decision systems.

**Timescale** Usability of a C2 support system like our DSS also includes fitting the timescale of the decision making process (Hazen, 2011). Running our simulation scenario can be done in around 10 minutes real-time, but the simulator can also run a lot faster. Retrieving the results in our DSS nearly only requires the simulator to have run the scenario since the postprocessing of the MOEs does not take more than a couple of seconds. It remains to be seen whether setting up the scenario, the systems and pushing the commands down to the line takes too much time, but the time taken by our agents and the DSS is so little that it shall definitely not be the driving force behind this requirement. Indeed, setting up the agent goes automatically as well, as the agents are created based on the provided MSDL. Moreover, the agents automatically process the C-BML order in a short amount of time and no human input is required during this process.

**Expert knowledge** Expert knowledge developed by other commanders should be provided to the simulation according to Hazen (2011). We have provided expert knowledge as developed by RNLA commanders to our agents in the form of the Combat Instruction Sets containing doctrine. As this knowledge evolves, engineers can adjust the agents and provide them with new beliefs, goals and plans to mirror this. Any change in this expert knowledge will of course represent itself in the DSS as well, as all its results are based on the agents acting on the simulated battlefield. Any C2 user that then joins the simulation setup with its C2 system (using HLA) will be able to reuse this experiential knowledge by sending his orders to the agents. Of course, loads of expert knowledge will still have to be added, but the framework is now available. Once engineered and available within the military organisation, any commander could potentially use the simulation and tap the expert knowledge of the agents, diminishing the need for a large staff of experts during his plan analysis and thereby opening up opportunities for increased autonomy on lower levels in the military hierarchy and cost-reduction.

**Added value** Simulation has to provide added value to the C2 process (Hazen, 2011). In this project we have added some value in the form of the DSS. No longer does the C2 - Simulation interoperability project focus only on the connection of standards, now we also provide the C2



users with measures of effectiveness taken from the simulation. The decision support system provides a quick overview of results that are not intuitively gatherable from merely looking at the simulator or C2 system. Although the DSS only shows a few MOEs at the end of this project, it provides the framework for future expansion to provide the C2 user with all the information that is wanted. Hazen (2011) provides the idea of exploratory analysis of variations on the provided COA, which we have decided not to implement during this Master's project, but it was one of our ideas as well, as can be seen in Appendix A.

**Intuitive results** A DSS should provide results that are intuitively understandable by military operators, and are credible in known situations (Hazen, 2011). This requirement is close to our research question, as we wanted to create an intuitively understandable (usable) DSS with credible (realistic) results. According to the previously discussed desires from experts we have created results that are intuitive to understand because we used simple MOEs that are generally part of COA evaluation. Furthermore, we only provide a single value for these MOEs, such as fuel percentage, instead of values for each individual unit. This allows for a quick overview, as alternatively the commander would have had to abstract from the individuals to the whole COA on its own. To obtain credible results we have introduced increased CGF realism based on requirements from other researchers and doctrine from the military.

Another important requirement from Prelipcean et al. (2010) about the usability of the DSS is the natural interface. We have provided a natural interface with our DSS table based on the expert's ideas, as discussed previously. However, as in Lafond et al. (2010), the optimal natural interface that fits in the C2 process should be based on the cognitive functions that benefit the most from it. We have seen that in CASA (Hanna et al., 2005), the DSS was extremely well-founded, but its interface made it less usable. One of their problems was that there was no filtering of information, thus creating a cluttered interface, the opposite of intuitive results.

**COA analysis** According to Prelipcean et al. (2010), the following set of functions would be needed in a DSS for COA analysis: description of the event, development / description of the possible COAs, identifications of criteria for evaluation, the actual evaluation of these COAs according to these criteria, analysis and comparison afterwards. Description of the event includes assumptions about the enemy and friendly forces, and thus it is basically equal to the standards for event description we use, called MSDL and C-BML. Since this project, MSDL and C-BML have been in use for the description of the scenario.

- *Development/description of the COA*

This, in this project, is exactly the same as it is during an actual battle or training because the same C2 system is connected to the simulator. Furthermore, the COA is sent with the C-BML standard towards our agents, providing the optimal ability for COA description and development. It should only be noticed that not all COAs are possible for the agent to execute of course. We have only implemented a small subset of the possible doctrine as time and manpower allowed. For a fully functional system, some more years of engineering would be required.

- *Identification of criteria to be used in the evaluation process*

Another related requirement from Prelicean et al. (2010) was to allow the staff to choose other criteria interactively after first providing our selection. This has already been discussed in previous Section 7.1.1, where we indicate how we can see in the results that criteria can be chosen. Our chosen criteria also match future work from Hanna et al. (2005), where they state that munitions and fuel expended could be a part of the COA scoring.

- *Evaluation of the COAs according to the selected criteria*

This is our main selling point. The DSS will provide results for all the chosen MOEs as can be seen in Figure 6.4 from the previous chapter.

- *Analysis and comparison of the COAs*

As stated in the previous chapter, comparing COAs is a bit more difficult in the current setup, because the agents should also be reset before acting upon a new COA. Since the DSS is linked to our Information Manager agent, it will also be reset. One possibility is shown in Figure 6.5, where the agents are started twice, providing two DSS windows for two different simulation runs. However, as shown in Figure 6.2b, the table does provide opportunity for comparison if there would be two different COAs in there. Each MOE column can be sorted and the columns can be switched around, essentially allowing full freedom for the user to rank COAs according to his or her wishes. Furthermore, the problem with resetting the agent and not the DSS should not be a difficult one to solve, allowing full advantage to be taken of the DSS.

**Lessons learned from other researcher's problems** In Section 2.3 we describe some other DSS projects and what their experiments resulted to. In Herbinet et al. (2010) we find problems with the lack of control and information. We also have some of their problems, like the limited number of orders and explanation from the agents, the information used by agents and a GUI (DSS) that is not designed for operational use. However, other problems that they discuss are solved in our setup: our system is designed for the battalion commander level, where the only interaction needed from the user is to provide orders to the battalion level, not to any lower level. Furthermore, we have used the 'train as you fight' paradigm throughout the experiment: the graphical symbols are all standardized NATO symbols and the C2 system in use is ISIS, as used by RNLA.

We show in our DSS prototype here that it is possible to connect C2 and simulation and provided added value by analysing the effect of the C2 order based on evaluation criteria. The connection to our agent is not yet optimal, as it inhibits restarting of the agents for a new order, but that could be solved by creating it as a standalone, similar to the C2Stub, which connects to the HLA bus and communicates to our agents in that way. This would also allow other users to create their own DSSs which use the same data but provide them with their own preferred interface. Besides that, according to all the recent literature on C2 - Simulation - DSS requirements, our new system uses most of the ideas that are currently flowing around about the C2 DSS. We allow creation of a COA, identification and selection of criteria and a quick

analysis in real time or faster of the agent's interpretation of this COA and provide the results in a simple and intuitive table view.

## 7.2 Realism

In Section 1.3.2, we discussed our initial research questions concerning realism in the simulator. The main question being: "How can we make the simulation more realistic?".

To get this realism, we have taken the expert knowledge of RNLA military doctrine, that was stored in the documents we refer to as Combat Instruction Sets (CISs). These documents describe how the RNLA would perform certain orders realistically. By implementing the behaviour set out in these CISs, we have implemented realistic behaviour in the agents and simulator and thereby provide the missing link in the chain of commander's intent, C-BML, agent reasoning and simulator behaviour. Furthermore, the increased realism in the simulator also increases the realism of the results in the DSS because they are directly linked.

**C-BML** One of the reasons for this project was to support C-BML, the standardized language for commands from humans to simulators. Because of C-BML, all thoughts and intentions of the commander should be translatable to machine language. This should allow the simulator to do that which the commander requires of it, so it provides an extra layer of realism. Now note that the other side of this medallion is that the simulator actually has to understand how to perform these C-BML commands. In the previous project from Bronkers, the Command Agents were set up, creating a framework in which C-BML orders could be translated from the high level to the low level. However, the C-BML orders implemented were not realistically translated, so the behaviour of the simulator was still not what would be intended by the commander. To solve this, our project injected a dose of realism in the translation of some C-BML orders to simulator behaviour.

### 7.2.1 Level of realism

Note that, while we might want to get the results closest to the truth for our DSS, complete realism on all levels is practically unattainable. Simply because of the hours of work and money that would have to be put into a project to achieve this. Instead, a simulation's desired realism has to be based on the needs of the user. A commander of a battalion will not need, nor want, to know if a soldier moves around an object past the left side or the right side. As such, knowing the audience of the simulation is vital to the success of this simulation. Our level of realism was tied to the company level as per the project's description.

Increase of realism in the simulator depends on the chosen domain, and ours was the domain of RNLA commanders providing orders on the company level. So, we found our answer to the realism research question in the application of the RNLA's company level military doctrine to the agent's behaviour. We have provided behaviour to the simulation that was not there before on this level, because the orders the simulator could carry out were restricted to the lower platoon level.

## 7.2.2 Requirements

The answer to our question of how to make the simulation more realistic, lies in the CGF behaviour. As such, we have conducted research into CGF requirements in Section 2.2.2.1. In this section we shall describe what requirements we see accomplished in our project, along the lines layed down by Figure 2.5 from Abdellaoui et al. (2009).

### 7.2.2.1 Architecture

As can be witnessed in Figure 4.6, we have injected the three major C2-Simulation interoperability standards into our system. C-BML is used to transform the C2 command into a language all systems can understand, HLA is used to transfer communication such as C-BML between all systems and MSDL is the latest addition that sets up the scenarios for all systems.

### 7.2.2.2 Autonomous Operation

**Deliberate** Deliberate behaviour is the main sign of intelligence and was sorely lacking in the Command Agents we started with. The order provided by a commander to his company is open to a lot of interpretation or autonomous deliberate decision making. With our phased-version of the Assault, that can even be interrupted for a blocking position, we have shown a great example of the possibilities of such deliberate autonomous behaviour in the Command Agents.

**Reactive** The other part of autonomous behaviour consists of reactive behaviour. One can mindlessly follow a plan, but certainly in military operations, reactivity is key to victory, and to realism. Of course, internally the agents all exhibit reactivity, since they respond with certain plans to orders and their own beliefs, but with autonomous reactive behaviour we mean actions of CGF in the simulator, based on other activities in the simulator. We have provided an example of showing reactive behaviour by implementing the Blocking Position CIS. As long as the enemy is confined to known locations, this CIS will not trigger any behaviour. Yet when enemies are advancing from unexpected locations, the Battalion agent's reactive side triggers and commands the nearest Company agent to apply a Blocking Position.

However, more reactivity is still required for more realism. In Appendix A, we have described some of the other realistic behaviour that our AI should exhibit. For this project, we have chosen to work with doctrine, which focusses on realistic deliberate proactive behaviour, so a lot of ground still has to be covered on the reactive part. This includes realistic reactive behaviour towards the terrain, e.g. a company cannot move into a line formation if there is not enough room, and towards the enemy, e.g. when fired upon the agents should react appropriately.

### 7.2.2.3 Realistic Behaviour

**Doctrine** Doctrine is a requirement for correct behaviour for our CGF. Without doctrine, the simulator cannot show behaviour that is realistic for the military. Doctrine is the rules of behaviour under certain conditions for the military personnel. Our increase in realism has of course been the implementation of this doctrine, from the well-discussed Combat Instruction Sets.

**Motivational** These include stress levels, survival instinct, moral motivations (Brandolini et al., 2004) and emotions (Tidhar et al., 1999), which all impact the decisions made by low-level entities. However, one could argue that the low-level decisions such as these motivational ones are to be made by the simulator, as our Command Agents keep themselves at and above the platoon commander level. Instead, more interesting might be an increased prediction capability, as this could provide all levels of agents with a much greater level of reactivity.

#### 7.2.2.4 Organization

**Sociality, Military Hierarchy** This section of organization has not changed much in this project, since we already portrayed such realistic behaviour at the start. Our lower level agents will not act on their own accord, only when ordered from above. With the blocking position order, we do show the possibility to interrupt lower level agents. Furthermore, by moving our units in formations, the planning of routes for individual tanks has seized, which adds extra enforcement of this military hierarchy.

**Interaction** Interaction between the agents has increased vertically, since the Company agent now commands the Platoon agents multiple times with the Assault CIS, where previously only the Seize command showed more than a one-on-one interaction between agents. Also, the reactivity of the Blocking Position CIS from the Battalion agent adds a level of interaction there where there was only communication of orders before. Now, the Battalion agent will hold the Company agent in a Blocking Position for as long as it wishes, but in the end releases it to continue its previous job. As of yet there is no horizontal interaction, in fact agents are not even aware that there are agents of their own type active, as such there is no interaction between two Platoon agents or two Company agents even though the formations seem to indicate such.

**Coordination** With our implementation of the Combat Instruction Set *Assault on enemy position*, we have shown a coordination between platoons that could not have been there without our company level agent. The different phases of the assault depended on the successful execution of the lower platoon level behaviour, but it also required the coordination and centralized control of the company level. Only when all platoons have successfully accomplished the actions set out for them in one phase, can the company carry on to the next phase. Where currently correct behaviour is automatically applied by our agents based on a single order, previously all the different subactions required for an assault would have to be provided in succession by the user of the system. Just as a commander will not waste valuable time on the battlefield explaining the action steps to the lowest levels, so has this need been erased in the simulation by our behavioural injections to these agents.

**Cooperation** As stated in the Interaction paragraph, there is no cooperation implemented as of yet. All activity is initiated top-down, which is a form of coordination and interaction but not cooperation.

**Communication, Military Reports, Feedback** In the communication and feedback section, the biggest change is the new decision support system. Internally, the communication between agents has changed little. However, the decision support system takes our military reports to a whole new level, as our agents now provide reports specifically intended to support the commander in choosing what COA works best.

### 7.3 BDI agents for C2-Simulation interoperability - good approach?

As a sidenote during this project, we have had interaction with FFI Norway as they are creating their own version of the Command Agents with the Context-based Reasoning framework. For this reason we devote this short section to the pro's and con's of using BDI agents, specifically those created with Jadex.

#### Pro's

- Very adaptable and flexible architecture. In fact, one could probably create Context-based Reasoning agents using BDI agents.
- Reusable plans (e.g. Move into formation, which is used for all the CISs we have implemented).
- A lot of support and expertise available for BDI agents. An example is the Jadex software system we could apply.

#### Con's

- The behaviour of the agents quickly gets complicated, which is also reflected in diagrams we have created. The inner-workings become difficult to explain.
- Communication between agents is not optimal. Sharing of knowledge between agents in a multi-agent system is critical, yet communicating beliefs to each other is cumbersome in Jadex. Other BDI implementations might have solved this problem though.
- The Jadex architecture and the Command Agents have a mismatch in capabilities. Performgoals in Jadex are supposed to be successful if a plan has successfully completed. However, when a plan of a Company agent completes, this probably means an order is sent to a Platoon agent, which does not say anything yet about what the result of this plan will be in the simulator. As such, goals might be prematurely successful, a problem we encountered in our TacticalRoadMarch plan.
- BDI is still relatively low-level, as we need to define all beliefs and how they trigger plans and what happens in the plans and what actions do, et cetera. It would be more accessible for non-expert engineers if a drawing of a plan or a UML diagram could be converted to correct behaviour automatically. For example, during my presentation of the results to TNO employees, one asked me if Figure 5.2 was automatically turned into code

/ behaviour, while another went even further by saying he would prefer Figure 4.1 to be the only required implementation. In truth, even Figure 5.2 is already greatly abstracted from the detail of implementation required for our BDI agents to work with the simulator. We have noticed already how the CISs did not describe all behaviour, for example how does a company move from one formation to another? All this low level behaviour is required in our agents.

- Every behaviour has to be pre-planned or hard-coded in one of the agent's plans. This requires expert knowledge about every possible decision, which might not be available or desired. Learning, or improvising or generalizing are all not part of the standard BDI architecture.

In conclusion, BDI agents allow for a great amount of freedom and can implement almost any behaviour, but this naturally entails that all behaviour will be required in detail and on low levels. The more freedom, the greater the detail required and the greater detail required, the greater the time, effort and expertise required.

However, one could argue that there is currently no way to circumvent this problem, as this is not so much a problem concerning BDI but one concerning the connection and communication between agents and simulator. In the end, the simulator will have to receive the low-level tasks as our main problem is still the translation of orders from C2's high-level to the simulator's low-level.

Yet, once this connecting layer is provided, perhaps a scripting language can be added on top of our Jadex BDI agents, capable of turning a diagram into implementation, creating new agent beliefs, communication between agents and a connection of pre-built plans. For examples we can see at a high-level scripting language for agents in Huang et al. (2005) and an example of turning UML diagrams into Java code in Usman and Nadeem (2009), so a combination of such features could very well be a useful route to take in the future.

So while it certainly seems possible to implement all desired expert behaviour into BDI agents, as is shown in this project, it also seems very plausible that there are other methods that would allow implementation with perhaps less detail and less freedom but at much greater speed or ease.

## 7.4 Future Work

### 7.4.1 Remark about level of realism and discussion with RNLA experts

Our final discussion with experts from RNLA's Simulation Center concerned this level of realism. While we had questions about the specific implementation of the provided CISs for our agents, their concern was about the level of realism and what we wanted to accomplish. In their view, it was not important how a platoon moved from one formation to another, because this level of realism was not what they needed. If no one is going to look at the simulation, because they only use its results, certain details of doctrine are already surplus. However, behaviour of company aggregates in the simulation is entirely based on the actions of its platoons. So this would indicate that for realistic company behaviour, realistic platoon behaviour is required.

For example, if a platoon of tanks does not move, no company formation will be formed. If a company formation is achieved, but one platoon moves somewhere else, the formation is broken again. Therefore, the implementation of company level behaviour in our agents is based on providing the platoon level with the correct behaviour. So to implement realistic behaviour on the company level, we need to have detailed descriptions of this behaviour on both the company and the platoon level.

On the same day of that final discussion with experts from SimCen, we talked to one of the military experts behind the CISs. Here, we once again did not receive answers to our questions, but got to see implementations of the platoon level behaviour in their simulator. This simulator requires an operator to command the platoons, but the command options are numerous. The detail of all these options is so great because soldiers are supposed to work together with the CGF from the simulator and think that the CGF are actually (operated by) real soldiers. According to them, anything above platoon level was just impossible to describe (and implement) in such detail as they did with their platoon level. While we already discarded a ton of information that was described in the Company-level CISs, compared to our implementation of them, they would have required a lot more information to get the level of realism required. For example, a gun turret of a tank is supposed to be pointed in a certain direction while in formation, e.g. not pointed at friendly units. This level of realism is needed when a soldier stares outside his virtual window and sees the gun turret of the CGF, but not when a commander is looking at the battlefield and pondering what order to provide his companies.

What this boils down to is that there is no more detailed description of realistic company behaviour available for us. However, what is not described might also not matter to the user. As such, assumptions can be made for the, unknown, more detailed levels.

However, implementing assumingly realistic behaviour is not very scientific, so for the future of this project, TNO should figure out for whom they are engineering these agents, what level of realism these users want to work with and how they are going to get the expert information required for this level and the levels below or above.

#### 7.4.2 DSS

A next step in the DSS engineering should be to allow the simulator and agents to be restarted, while the DSS table is maintained together with the data. This will allow multiple COAs to be shown in the same table. To get this done, the Information Manager will have to reset all but the table data. One way of doing this would be to store the table data in a database when closed and retrieve table data from a database when opened. Another way would be to send a message to the Information Manager to reset, upon which the agent will close all other agents and clear its own beliefbase except for some beliefs. A third way would be to store the DSS table on a different location, as is done with the C2Stub, and then let the Information Manager connect to this DSS table, if it is already opened.

Perhaps it might even be better to send the data for the table through HLA, allowing any DSS to receive the data if they are connected to the same HLA connection. This would bypass the whole problem with resetting the agents and the simulator.



**Multiple runs for one COA** Another important future update of the DSS is similar to what is described in Hanna et al. (2005). The COA should be scored over multiple runs so that COAs are not accepted because of rare results in one simulation run. Since the simulation is stochastic, a new run will provide different results. We have not done any experiments for this problem, so we cannot conclude anything yet, but the values we received in an Assault run could vary from 95% survival rate to 75% and for the enemy survival rate showed a similar  $\tilde{20}\%$  gap. Logically, this variance will be greater when a smaller army is used, as each destroyed tank will take up a much greater part of the %. However, those are outliers and most of the time the survival rate values are within 5% of each other. Furthermore, this variance shows only in survival rate, as the *fuel consumption* does not display any significant difference in different runs, and *completed* is binary and generally true in the end.

Perhaps the variation in results of one COA can even provide a metric of comparison in robustness. Similar to this idea is that the COA should be matched against a multitude of enemy COAs to see how robust it is against the enemy's plans.

**Cognitive aspects** As was heavily discussed in Section 2.3, the cognitive aspects of the C2 process will heavily influence the usability of all C2 decision support systems. While we have tried to match some cognitive aspects in our DSS, experiments with the end-users are needed for further development of the DSS, to see where improvements are to be made and how to best show results from the simulator runs of multiple COAs.

#### 7.4.3 CGF requirements

Another interesting connection concerns the requirement, from Tidhar et al. (1999), that CGF should be able to predict the intention of opposing forces. In (Bosse et al., 2011), theory of mind capabilities for BDI agents are researched, using a.o. prediction of the intention of other agents.

## Chapter 8

# Conclusions

In this study we wanted to find out how a useful decision support system could be created for military planning, that was based on our C2 and simulation connecting Command Agents. This problem was quickly divided into two disparate subjects: the usability of the decision support system's user interface and the realism of its results. For initial direction, we consulted related studies on usability requirements of a decision support system in the C2 domain and on realism requirements for the simulator's computer generated forces.

**Usability** From the usability requirements, we found that a decision support system should ever stay a support system, never a replacement system. Furthermore, both the decision support system and its results should be intuitive, quick and easy to use and, most importantly, fit into the decision making process. The decision making process in Command & Control is centered around the creation, analysis and comparison of courses of action, plans that hopefully lead to the accomplishment of the mission. While the creation of a course of action is a creative process that no current computer can or should emulate, a simulator fits naturally in the analyser role, setting the stage for a decision support system to help the commanding staff compare their courses of action based on the simulator's analyses. To stay supportive, the military staff should be able to provide its own course of action and its preferred criteria on which evaluation of this course of action should be based. This concept of our decision support system was also discussed with experts from the Royal Netherlands Army's Simulation Center, who thereby provided us with a set of measures of effectiveness on which a commander might analyse a course of action and a green light for our vision of a decision support system as a table with these measures as columns and courses of action as rows. By allowing the user to add or remove these columns containing the measures of effectiveness, we allow selection of analysis and comparison criteria. Moreover, the staff can add a new row to the table for every new course of action that they want to compare. By connecting this table to our Command Agents through the newly created Information Manager, tasked with retrieving the measures of effectiveness from the simulation, we provide a usable decision support system for military planning, while our agent-powered simulation works on the background to provide the analysis of the courses of action.

**Realism** As stated before, the analysis of a course of action is only useful if the analyser, our agent-powered simulator, provides a realistic environment. Influencing this realism are the simulator's models, our agents providing the interpretation of the commander's orders and the scenario in which both are set up.

To increase the realism shown by our agents, or rather by the simulator's computer generated forces, we turned to the related literature on this matter. According to Abdellaoui et al. (2009), the main requirements for the simulator entities' AI are deliberate and reactive behaviour, synergy from military organization, architecture flexibility, ability to learn from experiences and exhibiting correct & realistic behaviour. The latter is gained by physical modeling, a feature of the simulator, and military doctrine, expert knowledge about realistic behaviour sorely lacking from our Command Agents. Therefore, we turned to the Royal Netherlands Army and were provided with a set of Combat Instruction Sets (CISs), documents containing Dutch military doctrine. During this project we have implemented a subset of these CISs, each exhibiting new behaviour related to the entity requirements: Assault an enemy position; Hasty

occupation of a blocking position; Tactical road march and the Line and Column formations. The implementation of all this doctrine required a great increase in the available in- and output from the simulator, made possible by adjusting the communication ontology of all systems and the plugin of the simulator.

Assault an enemy position (or Assault) is a CIS that provides our agents with a great deal of deliberate behaviour and realistic organization. The Assault can be seen as the expert, or realistic, version of the previously implemented Attack. Instead of a single movement and setting of the engagement rules, the Assault essentially consists of 4 distinct phases that have to be coordinated within the company and platoons that are executing it. The management of these phases is done by our Company agents without the need for any human intervention, as it deliberately orders its subordinate platoons from one location and phase to the next.

Hasty occupation of a blocking position (or Blocking Position) is the CIS most notable for the reactive behaviour it instills in our agents. The Blocking Position is a formation of platoons set up by their company, in reaction to an unexpected enemy attack, in order to block this enemy from moving any further. With the addition of this doctrine to the Command Agents, we have created a reactivity that was not previously available, highlighted even more so when it is applied during the Assault. In our results we can see how the Battalion agent can order the assaulting Company agent to interrupt its Assault, apply a Blocking Position to suppress an unexpected enemy, and finally continue with the Assault's next applicable phase.

The Tactical road march and the Line and Column formations are essentially organization doctrines, as the platoons and/or companies are ordered into predefined formations, where the chosen offsets from each other of lower level entities provide emergent tactical benefits to the military group as a whole. This provides our Command Agents with most of the organization requirements put forward in the literature.

For the scenario setup we have turned to another SISO standard, MSDL or Military Scenario Definition Language, which allows definition of the military scenario in XML so that all simulators and C2 information systems can be initiated with the same scenario. Furthermore, we have used MSDL, which also contains the units used in a scenario, to automatically set up the correct amount and type of agents in our multi-agent system.

*Audaces Fortuna iuvat* (Virgil, 20 B.C.), or fortune favours the bold, as even the realism of the simulator's models has increased during this project. A new version of VR-Forces, version 4, has been released, providing, among other improvements, 3D models. Together with the other discussed improvements, we can only conclude that we have shown how to make the simulation results, and thereby the decision support system's results, more realistic.

**Future Work** We have shown improvements in architecture, autonomous operation, organization and realism, which together make up the requirements from Abdellaoui et al. (2009), save for learning. As such, future work on these computer generated forces' realism should focus on the ability to learn from experience. Beyond that, there are also other realism qualities in the literature that have not yet been touch upon in this project: the motivational requirements. These include stress levels, survival instinct, moral motivations (Brandolini et al., 2004) and emotions (Tidhar et al., 1999), which all impact the decisions made by low-level entities. However, one could argue that the low-level decisions such as these motivational ones are to

be made by the simulator, as our Command Agents keep themselves at and above the platoon commander level. Instead, more interesting might be an increased prediction capability, as this could provide all levels of agents with a much greater level of reactivity.

Of course, this study of the usability of a decision support system in the Command & Control domain cannot be viewed as conclusive. Though we have shown how an integration of this system with simulation and agent technologies allows for evidence-based wargaming and support in the critical comparison of courses of action section of the military decision making process, all this work really begs the question: can the created DSS be usable for a military commander? We have not provided empirical studies into the cognitive usability aspects inherent to decision support systems. As such, one of the most important future lines of research is to set up experiments to measure how such a decision support system best fits in with the cognitive side of the military decision making process, to determine what will have to be adjusted before such a prototype can be considered to be an asset to the commander's decision making.

However, if research and engineering efforts in these subjects continue, simulated-based decision support systems will most assuredly play a major role in the future preparation and execution of warfare.

# Bibliography

- Abdellaoui, N., Taylor, A. and Parkinson, G. (2009), Comparative analysis of computer generated forces' artificial intelligence, *in* 'NATO RTO Modelling and Simulation Group Symposium Proceedings, RTO-MP-MSG-069'.
- URL:** <http://ftp.rta.nato.int/public//PubFullText/RTO/MP/RTO-MP-MSG-069//MP-MSG-069-02.doc>
- Bosse, T., Memon, Z. A. and Treur, J. (2011), 'A recursive bdi agent model for theory of mind and its applications', *Applied Artificial Intelligence* **25**, 1–44.
- URL:** <http://dx.doi.org/10.1080/08839514.2010.529259>
- Brandolini, M., Rocca, A., Bruzzone, A. G., Briano, C. and Petrova, P. (2004), Poly-functional intelligent agents for computer generated forces, *in* 'Proceedings of the 36th conference on Winter simulation', WSC '04, Winter Simulation Conference, pp. 1045–1053.
- URL:** <http://dl.acm.org/citation.cfm?id=1161734.1161911>
- Bronkers, R. (2011), Command agents using bml for tactical decision support, Master's thesis, Vrije Universiteit Amsterdam.
- Bronkers, R., Henderson, H., de Reus, N., Alstad, A., Mevassvik, O. and Skogsrud, G. (2011), 'Battle management language capable computer generated forces'.
- Dahmann, J., Fujimoto, R. and Weatherly, R. (1997), The department of defense high level architecture, *in* 'Proceedings of the 29th conference on Winter simulation', IEEE Computer Society, pp. 142–149.
- Davidson, J. and Pogel, A. (2010), 'Tactical agent model requirements for m&s-based it -i c2 assessments', *The International C2 Journal* **4**(1).
- URL:** [http://www.dodccrp.org/files/IC2J\\_v4n1\\_05\\_Davidson.pdf](http://www.dodccrp.org/files/IC2J_v4n1_05_Davidson.pdf)
- Department of Defense, U. (2005), 'Mil-std-2525b'.
- URL:** [http://www.mapsyms.com/ms2525b\\_ch1\\_full.pdf](http://www.mapsyms.com/ms2525b_ch1_full.pdf)
- FM 101-5 Staff Organization and Operations* (1997), Headquarters, Department of the Army, Washington, DC.
- Hanna, J., Reaper, J., Cox, T. and Walter, M. (2005), Course of action simulation analysis, *in* 'Proceedings of the 10th International Command and Control Research and Technology Symposium (ICCRTS): The Future of C2'.

- Hazen, M. (2011), C2 challenges for modelling and simulation, *in* ‘Proceedings of the 16th International Command and Control Research and Technology Symposium (ICCRTS): Collective C2 in Multinational Civil-Military Operations’.
- Herbinet, J.-G., de Champs, P., Gautreau, B., Neugebauer, E., Thiel, A. and Khimeche, L. (2010), ‘C2&simulation coupling - lessons learnt from german & french c-bml experimentation’.
- Huang, Z., Eliëns, A. and Visser, C. (2005), ‘Step: a scripting language for embodied agents’, *Life-like Characters, Tools, Affective Functions and Applications* .
- Joint Education and Doctrine Division, J-7, Joint Staff, U.S. Department of Defense (2011), ‘Dod dictionary of military terms’.  
**URL:** [http://www.dtic.mil/doctrine/dod\\_dictionary/](http://www.dtic.mil/doctrine/dod_dictionary/)
- Lafond, D., Vachon, F., Rousseau, R. and Tremblay, S. (2010), ‘A cognitive and holistic approach to developing metrics for decision support in command and control’.
- Lucas, A. and Goss, S. (1999), The potential for intelligent software agents in defence simulation, *in* R. Evans, L. White, D. McMichael and L. Sciacca, eds, ‘Proceedings of Information Decision and Control 99’, Institute of Electrical and Electronic Engineers, Inc., Adelaide, Australia, pp. 579–583.  
**URL:** <http://www.eleceng.adelaide.edu.au/ieee/idc99/abstracts/lucas1.html>
- M & S Office, U.S. Army (2011), ‘Army modeling & simulation glossary’.  
**URL:** <http://www.ms.army.mil/library/glossary.html>
- Mcilroy, S. D., Mcilroy, D. and Heinze, C. (1996), Air combat tactics implementation in the smart whole air mission model, *in* ‘In Proceedings of the First International SimTecT Conference’.
- Prelipean, G., Boscoianu, M. and Moiescu, F. (2010), New ideas on the artificial intelligence support in military applications, *in* ‘Proceedings of the 9th WSEAS international conference on Artificial intelligence, knowledge engineering and data bases’, AIKED’10, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, pp. 34–39.  
**URL:** <http://dl.acm.org/citation.cfm?id=1808036.1808044>
- Rao, A., Lucas, A., Morley, D., Selvestrel, M. and Murray, G. (1992), Agent-oriented architecture for air combat simulation, *in* ‘Proceedings of Future Directions in Simulation Systems Workshop’.
- Rao, A. S. and Georgeff, M. P. (1991), Modeling rational agents within a bdi-architecture, *in* R. Allen and E. Sandewall, eds, ‘Second International Conference on Principles of Knowledge Representation and Reasoning’, Morgan Kaufmann, pp. 473–484.
- Rao, A. S. and Georgeff, M. P. (1995), Bdi agents: From theory to practice, *in* ‘Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)’, pp. 312–319.

RNLA (2011), 'Integrated staff information system (isis)'.

**URL:** [http://www.defensie.nl/english/army/materiel/communication\\_and\\_information\\_systems/information\\_systems/integrated\\_staff\\_information\\_system](http://www.defensie.nl/english/army/materiel/communication_and_information_systems/information_systems/integrated_staff_information_system)

Royal Netherlands Army, Ministry of Defence (2011), 'Organisation chart'.

**URL:** [http://www.defensie.nl/english/army/organisation/organisation\\_chart/](http://www.defensie.nl/english/army/organisation/organisation_chart/)

Sandercock, J., Papasimeon, M. and Heinze, C. (2004), An agent, a bot and a cgf walk into a bar..., in 'Proceedings of SimTecT 2004 Simulation Conference, Canberra, Australia'.

**URL:** <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.111.1594>

Shu-hsien and Liao (2000), 'Case-based decision support system: Architecture for simulating military command and control', *European Journal of Operational Research* **123**(3), 558 – 567.

**URL:** <http://www.sciencedirect.com/science/article/pii/S0377221799001095>

Sprague Jr, R. (1980), 'A framework for the development of decision support systems', *MIS quarterly* pp. 1–26.

Stolt, K. (2007), Sketch recognition for course of action diagrams, Master's thesis, Massachusetts Institute of Technology.

Surdu, J. and Kittka, K. (2008a), 'Deep green: Commander's tool for coa's concept', *Computing, Communications and Control Technologies: CCCT 2008* **29**.

Surdu, J. and Kittka, K. (2008b), The deep green concept, in 'Proceedings of the 2008 Spring simulation multiconference', Society for Computer Simulation International, pp. 623–631.

Tidhar, G., Heinze, C., Goss, S., Murray, G., Appla, D. and Lloyd, I. (1999), Using intelligent agents in military simulation or using agents intelligently; in 'Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence', AAAI '99/IAAI '99, American Association for Artificial Intelligence, Menlo Park, CA, USA, pp. 829–836.

**URL:** <http://dl.acm.org/citation.cfm?id=315149.315481>

U.S. Army (2011), 'Operational unit diagrams'.

**URL:** <http://www.army.mil/info/organization/unitsandcommands/oud/>

Usman, M. and Nadeem, A. (2009), 'Automatic generation of java code from uml diagrams using ujector', *International Journal of Software Engineering and Its Applications* **3**(2), 21–37.

van Doesburg, W. A., Heuvelink, A. and van den Broek, E. L. (2005), Tacop: a cognitive agent for a naval training simulation environment, in 'Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems', AAMAS '05, ACM, New York, NY, USA, pp. 34–41.

**URL:** <http://doi.acm.org/10.1145/1082473.1082801>



Vegetius Renatus, F. P. (390), *Epitoma Rei Militaris*, Vol. III.

**URL:** <http://www.thelatinlibrary.com/vegetius3.html>

Virgil (20 B.C.), *Aeneid*, Vol. X.

**URL:** <http://www.thelatinlibrary.com/vergil/aen10.shtml>

VT MÄK (2011a), ‘Artificial intelligence behavior modeling - b-have’.

**URL:** <http://www.mak.com/products/simulate/artificial-intelligence-behavior-modeling.html>

VT MÄK (2011b), ‘Computer generated forces - vr-forces’.

**URL:** <http://www.mak.com/products/simulate/computer-generated-forces.html>

Wooldridge, M. and Jennings, N. R. (1995), ‘Intelligent agents: Theory and practice’, *The Knowledge Engineering Review* **10**(2), 115–152.

Appendix A

## AI Upgrade Possibilities

To support commanders in their decision making, we need a system capable of automating part of the course of action (COA) creation and COA analysis processes. To this end, the agent system connecting C2 system ISIS and simulator VR-Forces should be used to compare measures of effectiveness (MOE) of multiple routes, providing feedback to the commander about the simulated results. This should provide the commander with a quick overview of the possible benefits of each route. However, before we can provide a realistic overview, we need a realistic simulation.

We have identified three possibilities in this setting for increasing the realism:

- Agent Behaviour
- Path Planning
- Enemy Behaviour

There is a certain overlap between these three and in the end all will be needed for a realistic simulation. However, due to time constraints, the current project can only focus on (a subsection) of one.

## A.1 Agent Behaviour

Currently, the agents only act as order distributors, providing a simplified order from the commander to the units in VR-Forces. Instead, we would like to refine the behaviour of the agents (and the units in VR-Forces), to enable more autonomy. The subordinate commanders (agents) should provide intelligent implementations of the commander's order. To this end, extra knowledge of doctrine and formations can be added, allowing the units to react more realistically in the simulation. For example, when currently fired upon, a company will keep driving towards its goal while shooting back at the enemy. Instead, we might like to see the company backtrack to a previously safe position to devise a new plan, based on a.o. the enemy's expected power, current mission objectives and terrain properties. Furthermore, the new plan should depend on the commander's intent: is the goal to get to a location as fast as possible or to clear the whole area of enemy resistance. This change in behaviour would effect the MOEs of a chosen route in a more realistic manner, e.g. increasing the amount of time but decreasing the amount of casualties taken when enemy is encountered.

By making the units respond to the environment (more realistically), the possibility of wargaming within the simulation might also be enabled. For example, the user could add enemy artillery during the simulation, to see how that would impact the MOEs of the current route. Currently the agents would not respond to this, while we might want them to devise a new plan to stay out of reach of the artillery.

Documents designed for TACTIS are available that describe military doctrine on the company and platoon level, which can be turned into plans for the agents. Such a document is called a Combat Instruction Set (CIS). The agent will then reason on when to use what doctrine based on its beliefs about the environment and its own mission.

## A.2 Path Planning

A commander only provides a set of *waypoints* that subordinate units should traverse. The actual route taken by the units is decided on a lower level. Because we are using simulation for these lower levels, this means that a path planning algorithm is needed to plan a path for these units between the provided waypoints. Currently, path planning is done by the B-HAVE module in VR-Forces. However, B-HAVE does not exhibit all the behaviour we would like to see from *military* path planning. For example, planning the shortest path *while staying as far away from enemy forces as possible* is not a possibility at this moment, while we might like to include such tactical options in the simulator. Furthermore, this module plans for single units or platoons while a path is needed for the company level, to ensure correct tactical behaviour. In the current system, one platoon might take a different route on its own if B-HAVE concludes that this route is faster. It does not take into consideration that such decisions should be made on a company level. Therefore, we might need to develop a path planning module in our agents that will use tactically-interesting information from the environment to plan routes on the company level.

There are two options: we can either replace the path planning module B-HAVE by a new path planning algorithm, or our agents can restrict themselves to planning extra waypoints between those provided by the commander, essentially refining his command. The latter should allow the lower-level units to move, using B-HAVE, without concerning themselves with tactical knowledge, while the agents only concern themselves with tactical path planning, leaving the other details to B-HAVE. To this end, the environment should be abstracted to a set of waypoints and connections between these waypoints. By adding costs to these connections based on their length, a minimization algorithm is able to find the shortest path between two points. The fastest route could be computed using the maximum speed possible combined with the length of such connections. This would provide similar results to what B-HAVE provides. However, extra information can be added as weights to connections or waypoints. For example, the presence of enemy units might make travelling along a path more expensive the closer it is to these units. On the other hand, travelling through canopy might provide a positive bonus. Moreover, by raising path planning to the level of company agents, the algorithm can plan while keeping in mind the formation that units have to keep, allowing for a more realistic simulation. Other additions could include a.o. surrounding terrain (cover), lines of sight, maximum velocities, weather influences and weights based on the current mission objective.

Perhaps more importantly, dynamic replanning of the path is a possibility currently not available in B-HAVE. For example, if a bridge would be destroyed along the pre-planned path, simulated units will stop and await new orders from the user. Instead, replanning could automatically be done on the company level, allowing agents to respond adequately to new information.

## A.3 Enemy Behaviour

Another important factor for providing relevant DSS results is that the enemy behaviour / responses to the friendly COA are good approximations of the reality. One possibility is to also create agents for the enemy units, providing them with similar reasoning capabilities and

similar reactive capabilities as friendly forces. This way, expert knowledge of combat tactics is easily reused. Some of these plans might then be turned off to show that the enemy units are not as expertly trained in these tactics as the friendly units.

Appendix B

## Combat Instruction Sets (CISs)

Documents designed for TACTIS<sup>1</sup> are available that describe military doctrine on the company and platoon level, which can be turned into intelligent agent behaviour. Such a document is called a Combat Instruction Set (CIS). The agent will then reason on when to use what CIS based on its beliefs about the environment and its own mission. In a second discussion with the TNO SME, see Section 3.1, we decided upon a specific subset of these CISs, which will be shown next.

These are the Combat Instruction Sets used during this project:

- Assault

- Company agent: *01329 Assault an enemy position.doc*
- Company agent: *02329 Mechinfptn Assault an enemy position.doc*
- Company agent: *01328 Fire and movement1.doc*
- Company agent: *01324 Final phase of an attack.doc*
- Company agent: *01327 Consolidate and reorganize.doc*

- Blocking Position

- Company agent: *00414 Hasty occupation of a blocking position.doc*
- Platoon agent: *01314 Hasty occupation of a battle position1.doc*

- Formations

- Company agent: *00403 Tactical road march.doc*
- Company agent: *00404 Column Formation.doc*
- Company agent: *00406 Line Formation Traveling.doc*
- Platoon agent: *01303 Tactical road march.doc*
- Platoon agent: *01304 Column Formation Traveling.doc*
- Platoon agent: *01306 Line Formation Traveling.doc*

---

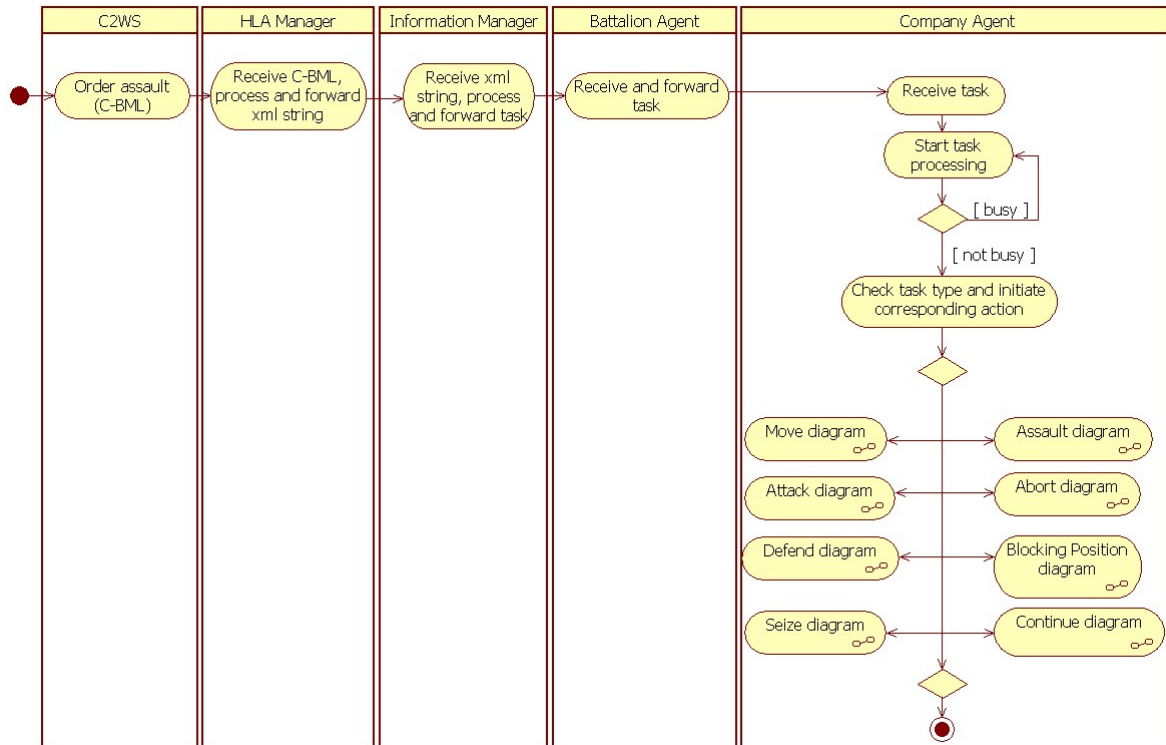
<sup>1</sup>Tactical Indoor Simulation from Thales. Tailored for combined forces instruction and training, see [http://www.thalesgroup.com/Portfolio/Defence/D3S\\_product\\_simu\\_tactis/?pid=1568](http://www.thalesgroup.com/Portfolio/Defence/D3S_product_simu_tactis/?pid=1568)

## Appendix C

# UML Diagrams



### C.1 Order processing



**Figure C.1:** A UML activity diagram showing how any order from a C2IS is processed.

In Figure C.1, we can see how an order from a C2IS is processed. First, the order is sent in C-BML and received as such by the HLA Manager. Then, it is forwarded to the Information Manager as a String, after which it is processed into a task object. The task then travels via the Battalion agent to the correct Company agent (because we only process company level orders). Finally at its destination, the task is processed if the agent is not currently busy with another task. If the agent is not busy, the type of the task will determine future actions.

### C.2 UML Activity Diagram - CIS Blocking Position

In Figures C.2 and C.3, we can see the discussed implementation as UML activity diagrams. Our blocking position is initiated in Figure C.2, along the green or red paths, when the simulator sends a unit status update to HLA. The HLA Manager always receives these units, stores it as an Equipment called eq and sends it to its listeners, i.e. the Information Manager. The Information Manager then receives eq, stores it in a detectedEquipment belief and continues with the processing of this detectedEquipment. Here the distinction is made between known and unknown units and enemy and friendly units. Enemy units will cause the Information

Manager to call Update Commander, which will inform the Battalion Agent, as can be seen in Figure C.3. The black path shows that friendly equipment are stored, their superior (platoon) agents are updated and that event listeners are registered for further information for our MOEs.

In Figure C.3, the green line shows us how the Battalion agent processes the received equipment (from the Information Manager). First it is checked whether the Equipment is not destroyed and that there are some expectedLocations, either from an initial setup, or stored from orders previously given. If so, distance is calculated, and when this distance is greater than expectedDistance, ABORTEXECUTINGTASK task is sent to the Company agent. This is later followed up with the BLOCKINGPOSITION task and the addition of the newly detected position to our expectedLocations belief.

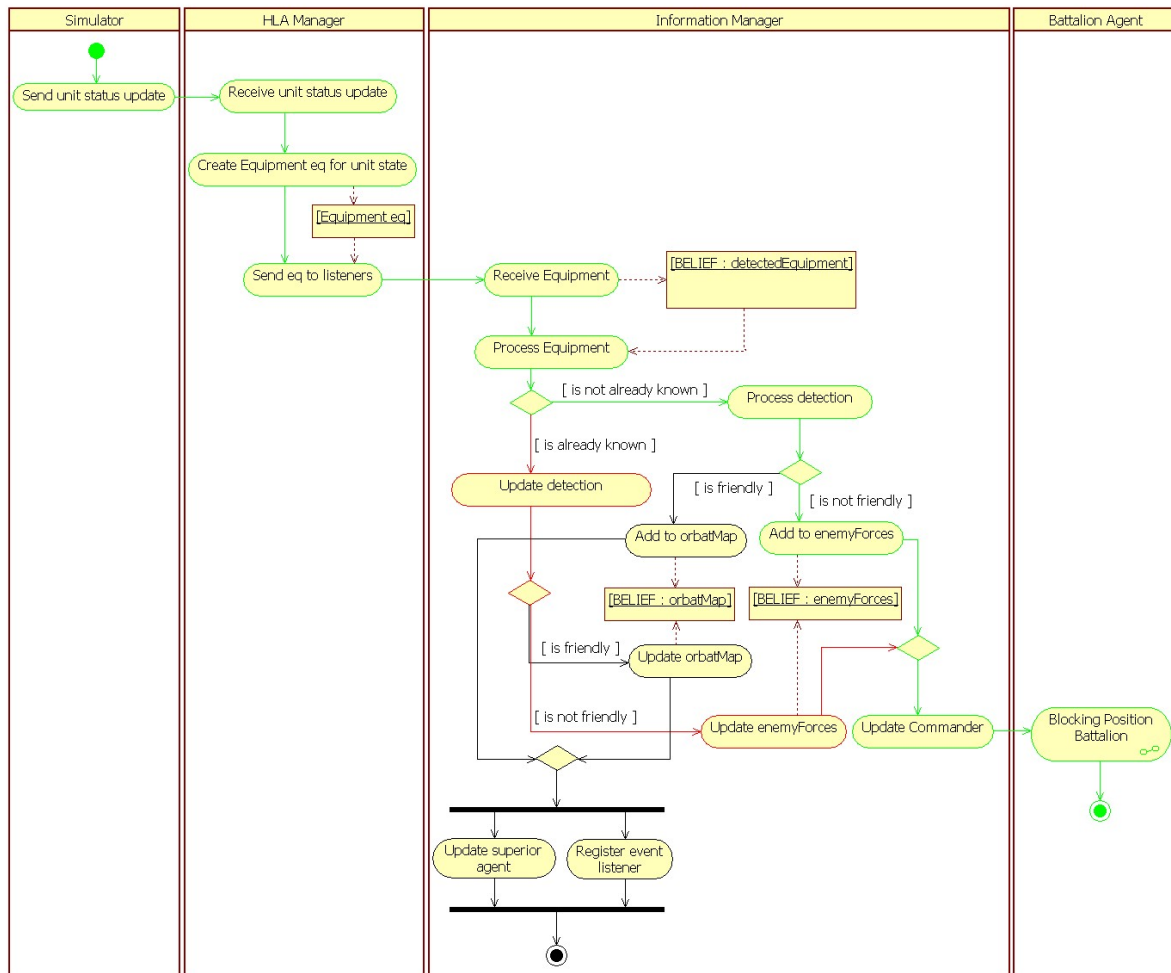
The red line shows how a new update of an enemy equipment (see Figure C.2), can stop the blocking position. This is the case if a destroyed Equipment is updated, which was ordered to be blocked earlier. In this case the blocking position is seized not by sending the ABORTEXECUTINGTASK task but the CONTINUE task.

### C.2.1 Unit Detection Update diagram

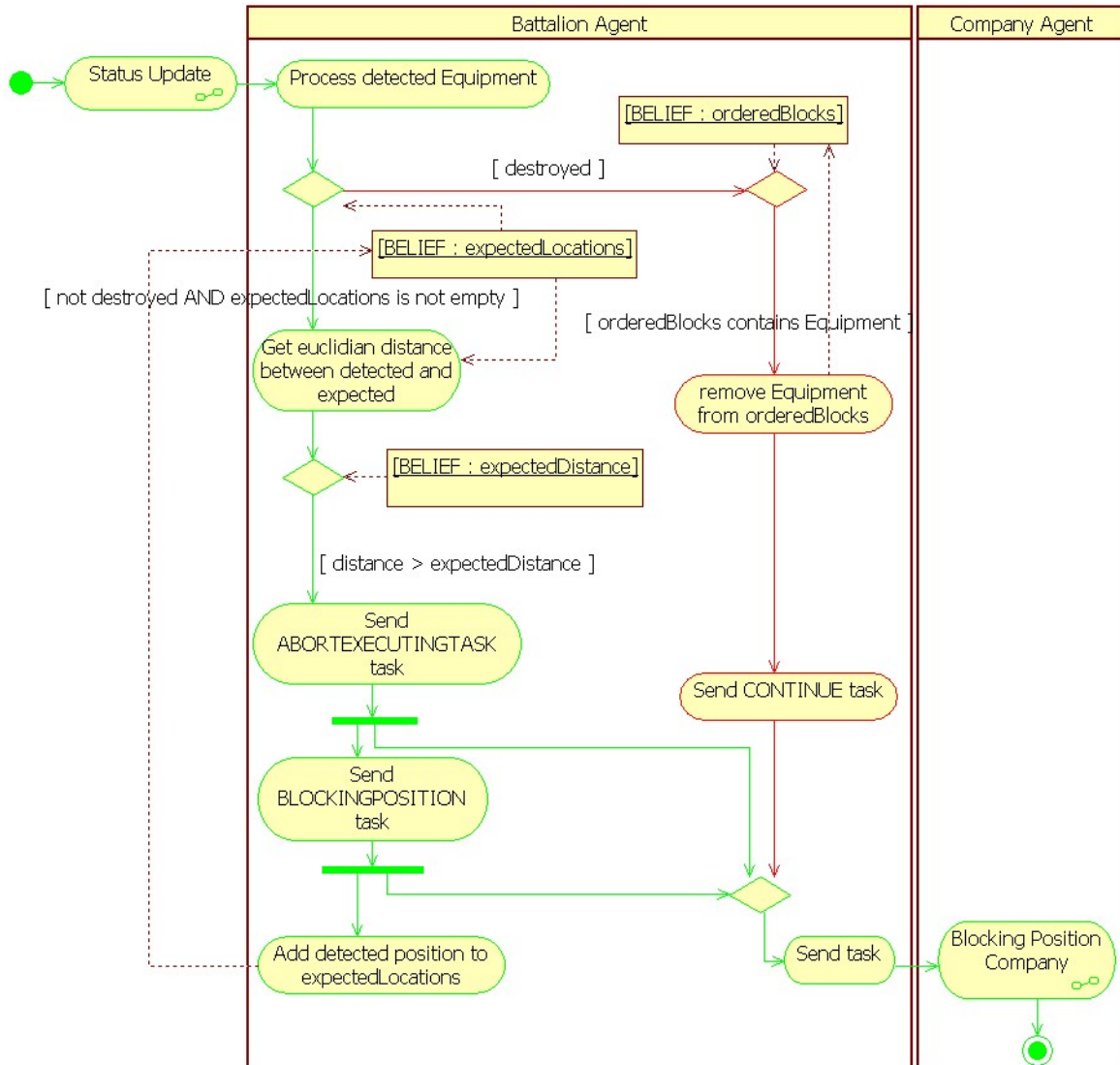
Figure C.2 shows the start of an extended activity diagram of the implementation of the CIS Blocking Position.

### C.2.2 Blocking Position Battalion diagram

Figure C.3 shows the extended activity diagram part that follows Figure C.2.



**Figure C.2:** An activity diagram showing the events leading up to a blocking position. The green path shows how an update from the simulator of a formerly unknown enemy equipment causes the Information Manager to Update Commander (Battalion Agent) about the new unit. The red path shows that known enemy equipments also cause for an Update Commander action. Black paths show what happens when friendly units are detected.



**Figure C.3:** The blocking position reaction from the Battalion agent shown in an UML activity diagram. The green path shows the decisions that lead up to ordering the company agent to apply a Blocking Position. The red path shows how a CONTINUE task is sent to the company agent, ending a previous Blocking Position.

## Appendix D

# Military Scenario Definition Language (MSDL)

## D.1 MSDL elements

A MSDL file consists of XML elements described in SISO's MSDL XML schema<sup>1</sup>. In this section we shall provide some excerpts from this schema, showing only those that are essential to the initiation of the Command Agents. XMLSpy 3 Version 4.1 was used by SISO to generate all the XML schema figures. An optional element is displayed in a rectangle with a dashed border, while the solid border displays the mandatory elements.

### D.1.1 msdl:MilitaryScenario Element

Figure D.1 shows the main structure of the MSDL file.

### D.1.2 msdl:Organizations

Figure D.2 shows the structure of the Organizations element available as child of the MilitaryScenario element.

### D.1.3 msdl:Units

Figure D.3 shows the structure of the Units element available as child of the Organizations element.

### D.1.4 msdl:Unit

Figure D.4 shows the structure of the Unit element available as child of the Units element.

### D.1.5 msdl:Equipment

Figure D.5 shows the structure of the Equipment element available as child of the Organizations element.

### D.1.6 msdl:EquipmentItem

Figure D.6 shows the structure of the EquipmentItem element available as child of the Equipment element.

---

<sup>1</sup>[http://www.sisostds.org/DigitalLibrary.aspx?Command=Core\\_Download&EntryId=30830](http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=30830)

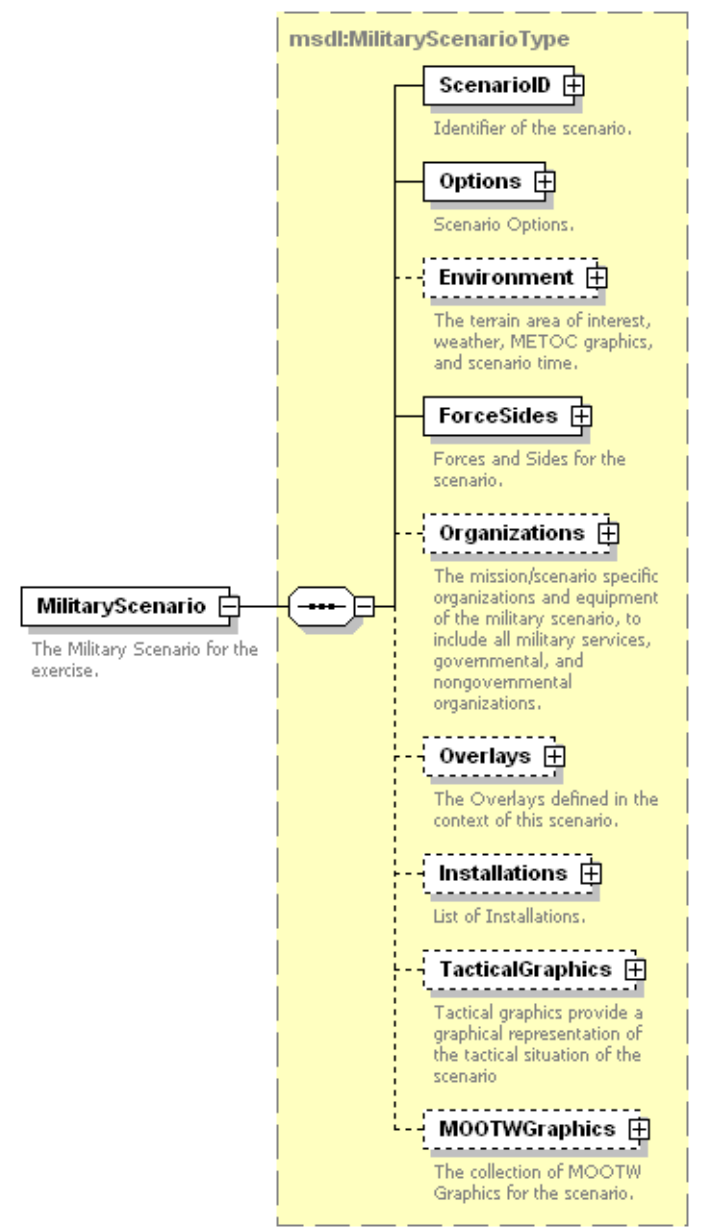
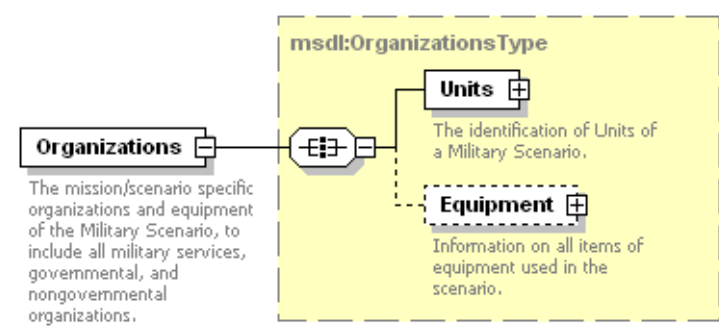
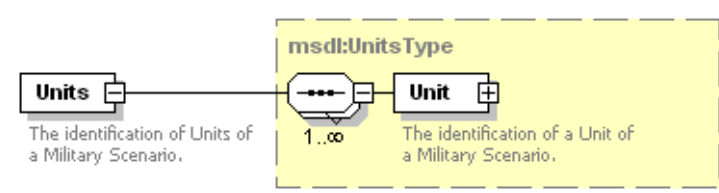


Figure D.1: *msdl: MilitaryScenario* Element Structure



**Figure D.2:** *msdl:Organizations Element Structure*



**Figure D.3:** *msdl:Units Element Structure*



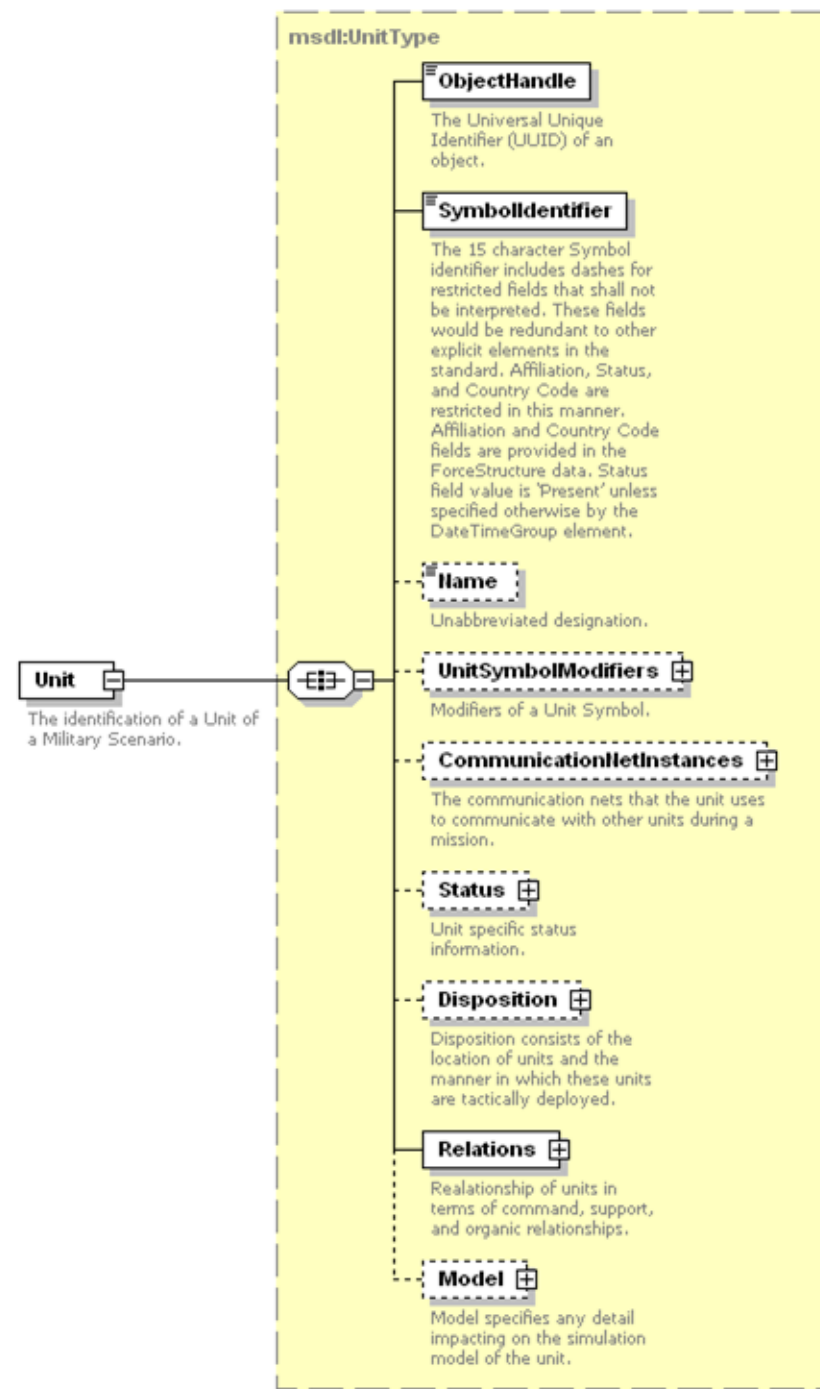
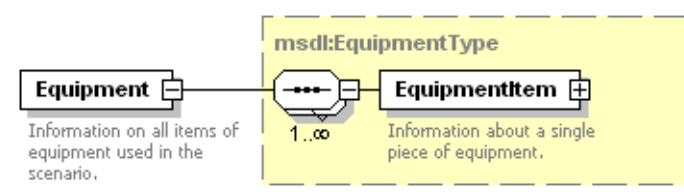


Figure D.4: *msdl:Unit* Element Structure



**Figure D.5:** *msdl:Equipment Element Structure*

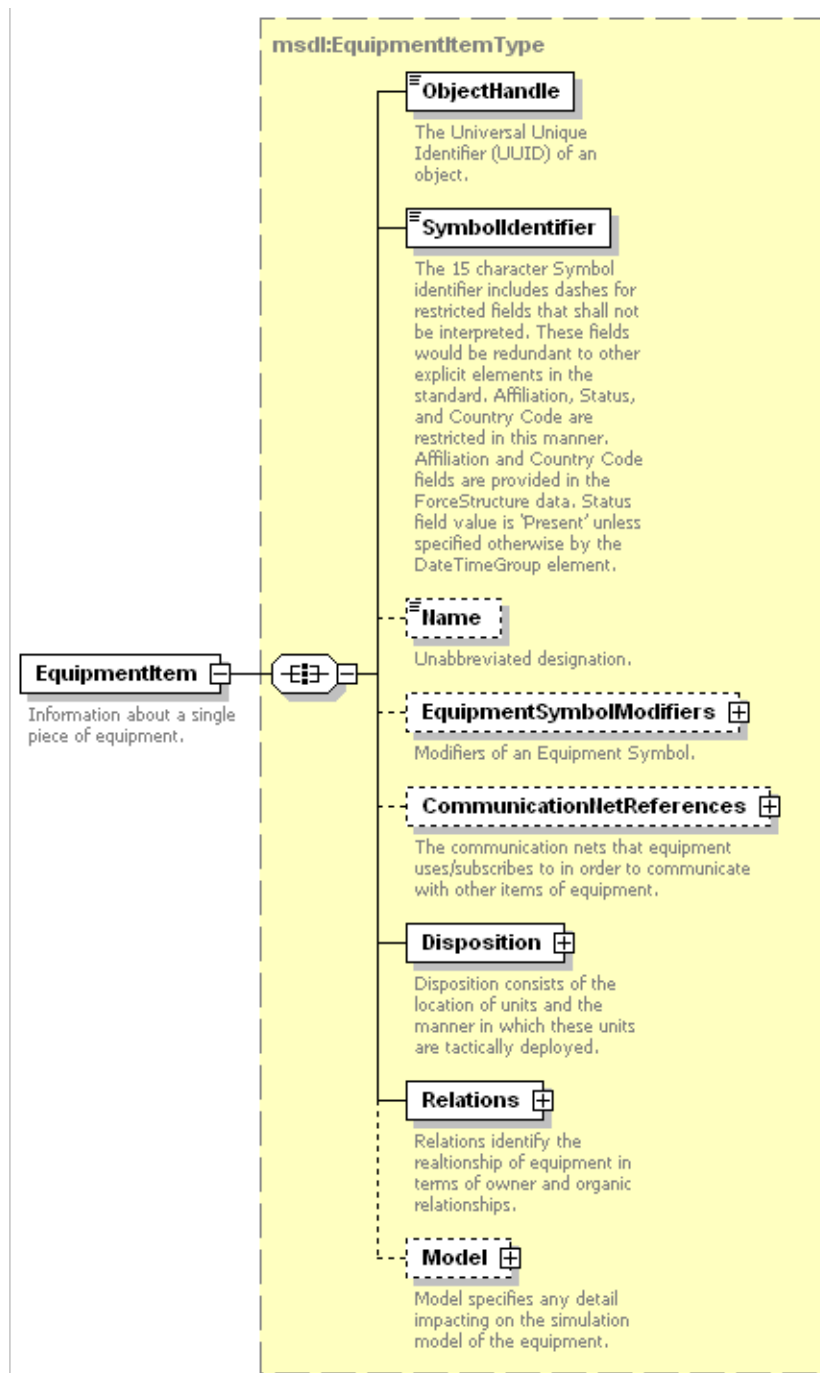


Figure D.6: *msdl:EquipmentItem Element Structure*

Appendix E

FOM

## E.1 FOM changes

The changes we made to the RPR-FOM are shown in Figure E.1, which is the overview of the `BML_evolved_module`. It was done to get the following functionality:

### Simulator output

- DetectionEvent
- EntityEvent
  - EntityInArea
  - EntityCrossedLine
  - EntityDestroyed
  - EntityFuel
  - EntityAmmunition

### Simulator input

- MoveIntoFormation
- AbortExecutingTask
- TurnToHeading
- FollowEntity
- SetHeading
- SetSpeed

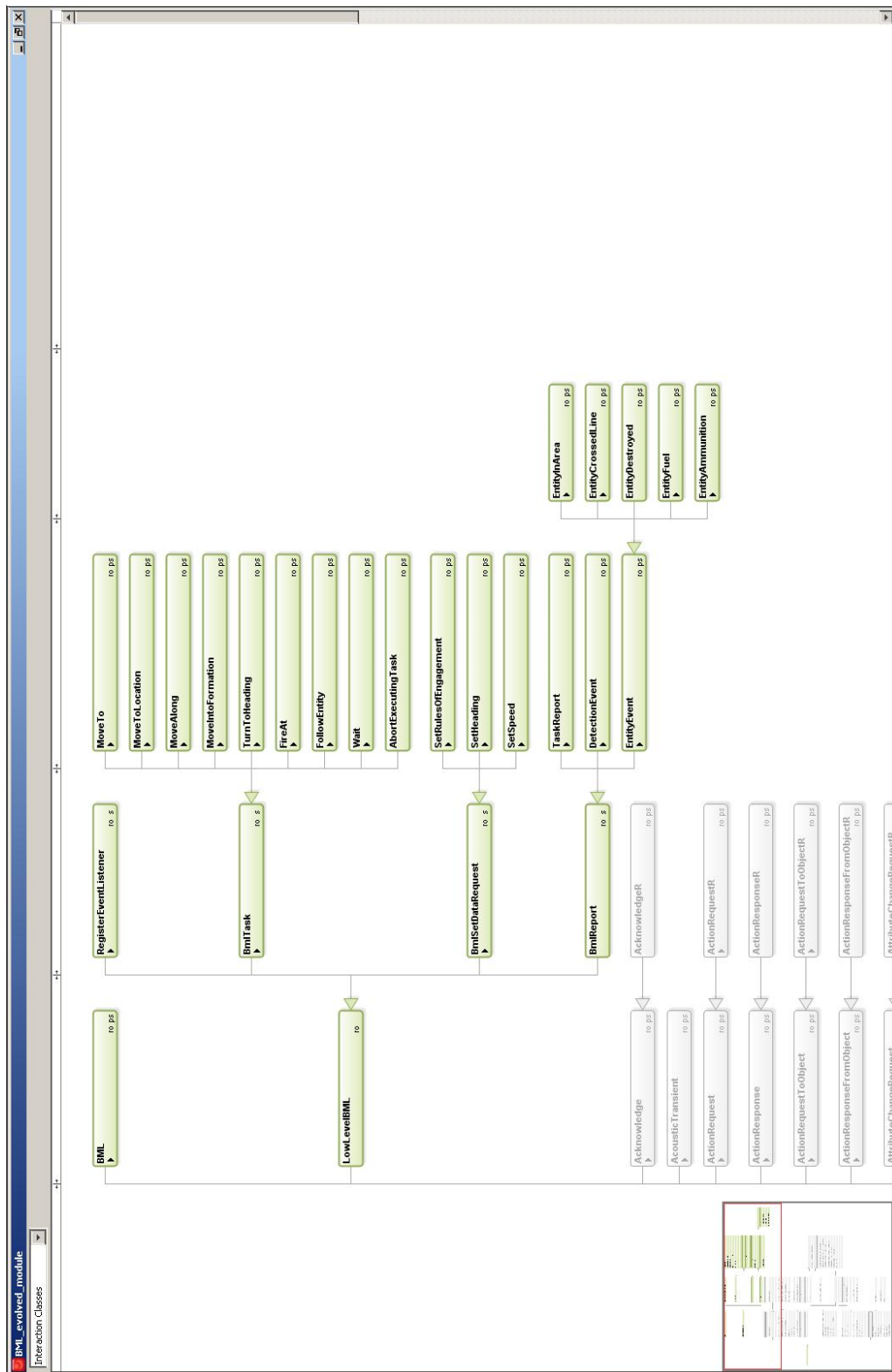


Figure E.1: HLA (Evolved) FOM module containing all the entries needed for this C2SIM project.