FFI-rapport 2013/01547

Autonomous battalion simulation for training and planning integrated with a command and control information system

Anders Alstad, Rikke Amilde Løvlid, Solveig Bruvoll
and Martin Norman Nielsen

FFI Forsvarets forskningsinstitutt

FFI-rapport 2013/01547

Autonomous battalion simulation for training and planning integrated with a command and control information system

Anders Alstad, Rikke Amilde Løvlid, Solveig Bruvoll and Martin Norman Nielsen

Norwegian Defence Research Establishment (FFI)

13 January 2014

FFI-rapport 2013/01547

1233

P: ISBN 978-82-464-2328-9
E: ISBN 978-82-464-2329-6

Keywords

Distribuert simulering

Modellering og simulering

Agenter

Kunstig intelligens

Approved by

Karsten Bråthen	Project Manager
Anders Eggen	Director

English summary

Current Command and Staff training uses simulation systems that consist of computer generated forces in combination with human operators. The human operators receive high level tasks (e.g. company level) as input, transform these into lower level tasking for subordinate units (platoon level and lower), and then they manually enter the more detailed sets of instructions into the simulation system. A challenge in the case of training is that the amount of resources required inhibits a high frequency of training events. The need for a large simulation supporting staff is even more problematic if simulations were to be used more during operations in planning or mission rehearsal, e.g. for what-if analysis. We are investigating how we can make a more autonomous simulation system, which interfaces Command and Control Information Systems (C2ISs) in a seamless way, minimizing the number of human operators.

In order to realize a seamless integration of a simulation system with a C2IS, an order made in the C2IS must be expressed in a standard, unambiguous language, which is interpretable by the simulation system. Also, to make the simulation system able to carry out higher level operations (e.g. battalion operations) autonomously, the simulated forces must have sufficient knowledge about tactics and doctrine.

In this report we describe the design and implementation of a first version of a demonstrator of a simulation system capable of autonomous simulation of battalion operations. The simulation system is integrated with a C2IS, which can be used to visually view and create orders, in addition to presenting ground truth and perceived truth. The simulation system is capable of receiving and executing orders created by the C2IS and providing reports back to the C2IS. In addition, the C2IS is used to define the ORder of BATle (ORBAT) and provide the initial positions for friendly units. The exchange of orders, reports and scenario definitions between the C2IS and the simulation system are expressed in Coalition Battle Management Language (C-BML) and Military Scenario Definition Language (MSDL), where C-BML is a standard under development for exchanging orders interpretable by machines and MSDL is a standard language for describing scenarios.

The simulation system consists of a Multi-Agent System (MAS) and a commercial off the shelf CGF system. Knowledge of higher level tactics and doctrine are implemented in the MAS, which is used to control the entities in the CGF system. The agents in the MAS are organized in a hierarchy and represent leaders and staff of military units. The behaviour model is based on the human behaviour modelling paradigm Context-Based Reasoning (CxBR). The implementation of the CxBR based MAS framework and the behaviour model implementation are documented in this report.

The simulation system integrated with a C2IS was demonstrated for subject matter experts (SMEs), and the response was generally positive. Feedback from SMEs, regarding possible applications for such a system, what the system should and should not do etc., is summarized in the report.

Sammendrag

Dagens simuleringsbaserte stabs- og ledertrening krever at operatører styrer simuleringen. Disse operatørene tar i mot kommandoer på høyere nivå, som de bryter ned til oppgaver på lavere nivå, som brukes til å styre datagenererte styrker. Nødvendigheten av ekstra personell i form av operatører begrenser hvor ofte det er mulig å trene. Dette gjør det også kostbart å bruke simulering til evaluering av handlemåter under operasjonsplanlegging. Vi jobber med å utvikle mer autonomt simuleringssystem. Dette systemet kan blant annet kommunisere med et kommando og kontroll informasjonssystem (K2IS) direkte, slik at mengden nødvendig personell minimeres.

For å kunne integrere et simuleringssystem med et K2IS, må ordren som lages i K2IS utrykkes i et standard, utvetydig og maskinleselig språk som simuleringssystemet forstår. I tillegg må simuleringssystemet være i stand til å forstå og utføre høyere nivå oppgaver, som betyr at de simulerte styrkene må ha tilstrekkelig kunnskap om standard taktikk og doktrine. Denne rapporten beskriver en første versjon av en demonstrator av et simuleringssystem som er i stand til å autonomt simulere en bataljonsoperasjon.

Simuleringssystemet er integrert med et K2IS som brukes til å se og definere ordre samt følge med på status til egne og fiendtlige styrker. Simuleringssystemet kan ta i mot og utføre ordre definert i K2IS og sende rapporter tilbake. K2IS brukes også til å definere oppdragsorganisasjon og startposisjoner for egne styrker. Utveksling av ordre, rapporter og senariodefinisjoner mellom K2IS og simuleringssystemet er utrykt i "Coalition Battle management Language" (C-BML) og "Military Scenario Definition Language" (MSDL). C-BML er en standard under utvikling for utveksling av maskinlesbare ordre og MSDL er en standard for å beskrive scenarier.

Simuleringssystemet består av et multiagentsystem sammen med et kommersielt tilgjengelig system for datagenererte styrker. Kunnskap om høyere nivå taktikk og doktrine er implementert i multiagentsystemet, som brukes til å kontrollere de datagenererte styrkene. Agentene i multiagentsystemet er organisert i et hierarki som representerer lederne med stab for de aktuelle militære enhetene. Kontekstbasert resonnering (CxBR) er brukt til å modellere adferden til agentene, og implementasjonen av dette CxBR-baserte agentrammeverket er hovedfokuset i denne rapporten.

Simuleringssystemet og hvordan det virker sammen med K2IS har blitt demonstrert for militære eksperter, og tilbakemeldingene var generelt positive. Tilbakemeldinger om mulige bruksområder, hva systemet bør gjøre og ikke bør gjøre osv. er oppsummert i slutten av denne rapporten.

Contents

1 Introduction 7

2 Objective and requirements 9

3 Applied technologies and applications 9

3.1 C-BML and MSDL 10

3.2 High level architecture 10

3.3 Intelligent agents 11

3.4 Context-based reasoning 11

3.5 VT MÄK VR-Forces and VR-Link 13

3.5.1 Entity models and behaviour 14

3.5.2 Terrain 14

3.5.3 Radio communication, perceived truth, blue force tracking 15

3.5.4 Exercise clock 15

3.5.5 B-HAVE 16

4 System design 17

4.1 The command and control information system 17

4.2 Communication between the C2IS and the MAS 18

4.3 The multi-agent system 19

4.4 The CGF system 20

4.5 Communication between the MAS and the CGF system 20

4.5.1 Low-level BML 20

4.5.2 Time management 20

4.6 Final system configuration 21

5 Multi-agent system framework 22

5.1 Requirements 22

5.2 Approaches investigated 23

5.3 Architecture and design 24

5.4 Implementation 26

5.4.1 Communication services 26

5.4.2 Scenario initialization 27

5.4.3 Agents 28

5.4.4 Order processing 30
5.4.5 Simulation execution 30
5.4.6 CxBR implementation 32
5.4.7 Splitting of routes 34

6 Configuration and extensions of VR-Forces 37
6.1 Low-level BML plug-in 37
6.1.1 Low-level BML plug-in design 38
6.1.2 Low-level BML message set definition 39
6.1.3 Simulation model improvements 40
6.2 Use and customization of models in VR-Forces 42
6.2.1 Entity models 42
6.2.2 Formations 43
6.2.3 B-HAVE 44

7 Experiment 45
7.1 CxBR behaviour model 45
7.2 Scenario in VR-Forces 45
7.2.1 The terrain in the scenario 47
7.2.2 Testing of routes and positions 48
7.2.3 Red forces and their behaviour 48

8 Results 49
8.1 Possible applications and desired functionality 49
8.2 Comments on the current behaviour 50

9 Conclusions and future work 50
9.1 Topics for discussion 51
9.2 Future work 51

Bibliography 55

1 Introduction

Current Command and Staff training uses simulation systems that consist of computer generated forces (CGF) in combination with human operators, so-called LOwer CONtrol operators (LOCONs). These LOCONs receive high level tasking (e.g. company level) as input, transform this into lower level tasking for subordinate units (platoon level and lower) and then they manually enter this more detailed set of instructions into a simulation system.

Training events are usually big events where a large number of LOCONs are required next to the instructor staff. Besides the transformation of tasking, the LOCONs and instructor staff take care of scenario initialization and trainee evaluation. Although the general idea is that these lower level operators also benefit from this work, it is usually more difficult to train multiple levels because of the generally different training goals for these levels.

A challenge in the case of training is that the amount of resources required inhibits a high frequency of training events. If the number of LOCONs could be reduced by (partial) automation of their job, this could greatly enhance the number of training events and consequently mission readiness.

Simulation systems can also be used during operations in planning or mission rehearsal, which even more inhibit the use of a large simulation support staff. For instance, in the planning phase of an operation, simulation systems can be used to do what-if analysis. In these circumstances the war-fighter requires faster than real-time simulation speed and the need for LOCONs should be as limited as possible.

The military simulation applications mentioned above would benefit from a capability that interfaces Command and Control Information Systems (C2ISs) with simulations in a seamless way, minimizing the number of LOCONs necessary. In order to realize this seamless integration, an order made in the C2IS must be expressed in a standard, unambiguous language that is interpretable by the simulation system, and the simulated forces must have sufficient knowledge about tactics and doctrine to carry out the operation.

Since 2005 FFI has participated in NATO science and technology research groups that have focused on developing a standard for a Coalition Battle Management Language (C-BML). C-BML defines a standardized language for exchanging orders, requests and reports between C2ISs, CGF systems and robotic forces [1]. Until 2012 FFI has participated in the C-BML standardization effort through working with the standard itself and through providing a C-BML capable C2IS. This C2IS capability has been created through the use of NORTaC-C2IS from Kongsberg Defence Systems (KDS).

However, existing COTS CGF systems covering the land domain are in general not capable of processing and simulating C-BML orders and requests. C-BML captures orders and requests in a Command and Control (C2) language that typically addresses units at company level and above.

This requires that a C-BML compliant simulation systems models battle command for higher level units.

In 2011 FFI and TNO started cooperating to create a C-BML capable CGF system in the framework of the Anglo-Netherlands-Norwegian Cooperation Program (ANNCP). The motivation was to enable autonomous simulation of orders created in a C2IS. In addition to providing experience with implementing C-BML simulation capabilities, this activity also supported experimentation with training without a large exercise staff and operational planning. The collaborative work with TNO is described in [2] and [3].

The collaboration between TNO and FFI has focused on creating a C-BML capability based on a common Commercial Of-The-Shelf (COTS) CGF system. Each nation has developed a Multi Agent System (MAS) that is used in conjunction with VT MÄK VR-Forces. These MASs have been designed to process higher-level orders (e.g. a battalion order) and decompose them into lower-level commands according to military tactics and doctrine. These low-level commands have then been sent to and simulated by VR-Forces, which in return has provided simulation state and low-level reports back to the MAS. Based on the received simulation state and low-level reports the MAS simulates tactical decision making, in addition to providing high-level C-BML reports back to the C2IS. VR-Forces combined with a MAS thus becomes a C-BML capable, more autonomous CGF system.

Parallel to the ANNCP cooperation with TNO, FFI has also collaborated with Professor Avelino J. Gonzalez, Director of the Intelligent Systems Laboratory at the University of Central Florida, to investigate the use of Context-Based Reasoning (CxBR) for modelling of tactical command and control and battle command. Our MAS has therefore been developed using the CxBR paradigm [4, 5, 6].

This report documents our effort of creating a demonstrator for autonomous simulation of battalion operations integrated with a C2IS. This simulation system consists of a COTS CGF system for low-level entity simulation combined with a MAS for higher-level tactical decision making. The main objective and requirements for this system are listed in section 2. Section 3 provides background information on the technologies used in the simulation system. An overview of the total simulation system together with information about communication solutions between the components are described in section 4. Section 5 goes into details about the implementation of the MAS framework, and section 6 summarizes the extensions and configurations done in VR-Forces. Information about the behaviour models we implemented in the MAS and the scenario we used to test the simulation system is provided in section 7 and section 8 summarizes the feedback we got after a demonstration for military experts. We conclude with topics for discussion and future work in section 9.

2 Objective and requirements

The main objective of the demonstrated simulation system is to create a C-BML capable CGF system capable of simulating Norwegian tactics and doctrine and by such enable experimentation with training without a large exercise staff and operational planning.

The requirements are as follows:

Scenario initialization: Scenarios in the simulation system shall be initialized through scenario initialization documents produced by and received from a C2IS.

Simulation of orders: The simulation system shall receive and simulate digitized orders produced by a C2IS.

Reporting of simulation state: Simulation state shall be sent as digitized reports to a C2IS.

Norwegian military doctrine: The execution of orders shall be simulated according to Norwegian military doctrine and tactics.

Autonomity: The demonstrated simulation system shall be able to simulate the battalion operation without humans in the loop.

C-BML: Digitized orders and reports exchanged between systems shall be expressed using C-BML [7]. More specific, the C-BML type Joint Battle Management Language (JBML) [8] shall be used as this is already supported by the FFI C2-gateway [9].

MSDL: Military Scenario Definition Language (MSDL) [10] shall be used to encode scenario initialization documents.

Multi-agent System: Military tactics and battle command shall be simulated using intelligent agents for representing the leaders and staff in a military organization. These intelligent agents shall be part of a MAS.

CxBR: CxBR shall be used to model the behaviour of the agents.

CGF system for entity simulation: Low-level entity simulation shall be left to a CGF system, because we want to concentrate on modelling higher level tactics and battle command, and because low-level entity simulation already is done well by existing CGF systems.

MAS independent of CGF system: The MAS shall be independent of the selected CGF system, such that exchange of the CGF system at a later time becomes a simple task. This requires a standardized language for communication between the MAS and the CGF system.

VR-Forces as CGF system: The CGF system for this version of the simulation system shall be VR-Forces from VT MÄK. This CGF system shall be used because both FFI and TNO currently are using it and because it easily can be extended with new functionality.

Java: The MAS shall be programmed in Java, since the group is familiar with this language and it is considered suitable for implementation of this type of system.

3 Applied technologies and applications

This section presents the technologies used and existing efforts in creating a C-BML capable CGF system.

3.1 C-BML and MSDL

Battle Management Language (BML) is defined as *"The unambiguous language used to command and control forces and equipment conducting military operations and to provide for situational awareness and a shared, common operational picture"* [1]. C-BML is an attempt to standardize digitized formats for orders, reports and requests for coalition operations. C-BML is being developed not only to support inter-operation among C2ISs and simulation systems, but also to describe the commander's intent in a way that war-fighters can understand and make use of. As such it will also be applicable to command and control systems and unmanned systems.

A C-BML order is based on the 5Ws, Who, What, When, Where, and Why. The most important information elements are *who is tasked* (in our case which company), *who issued the order* (the battalion commander), *what are the tasks*, including constraints telling when the task is to be executed (time or dependency on other tasks), and *control measures* informing and restricting the execution of tasks (boundary lines, axis of attack, phase lines, objective areas, routes, etc.).

MSDL is another language associated with C2-Simulation interoperability. Where C-BML is intended to express orders and reports, MSDL is a standard for expressing military scenarios independent of the application generating or using the scenario [10]. MSDL enables exchange of all or parts of scenarios between C2ISs, simulations and scenario developing applications. A scenario can include specifications of forces and sides, unit configurations and relationships, locations, weather etc.

Both C-BML and MSDL is defined using an eXtensible Markup Language (XML) schema, and there are efforts being made to ensure compatibility between these two languages [11].

3.2 High level architecture

To simulate large and complex system, it is often necessary to distribute parts of a simulation on several computers. These computers can be located at different geographical locations, but the result of the simulation should be the same as if they all were run on one computer. There are several technologies available for connecting distributed simulations. In military modelling and simulation it is common to use High Level Architecture (HLA) [12], which is why we have chosen this technology for the communication between the MAS and the CGF system.

HLA can be described as a component based middleware architecture for simulation. This kind of architecture is designed for dividing a simulation into smaller components and distributing these components on several computers. These components can be reused in different simulations by putting them together in new configurations, and they can be designed and implemented by experts in the particular area of expertise that the component is representing. An interface between the components (HLA) is needed for communication, and there must be a common interpretation of the data that are exchanged between the components.

The common interpretation of the data that are exchanged is defined in the Federation Object

Model (FOM). A FOM describes what is simulated and what kind of data can be exchanged. For military simulations, the exchanged data are typically aircraft, vessels, vehicles, sensors like radar, radio, radio messages, etc. A FOM also describes how the data that are exchanged are represented (in bits and bytes). The modelling and simulation projects at FFI have chosen the Real-time Platform Reference (RPR) FOM version 2.17d as a standard [13]. When using HLA, the simulation components are called federates. A federation is a collection of federates. While a FOM apply to the whole federation, each federate can also define each own Simulation Object Model (SOM). The SOM must be a subset of the federation's FOM.

The interaction between the federates is managed by a Run-Time Infrastructure (RTI). There are many different RTIs, and a comparison can be found in [14]. The latest HLA standard, IEEE 1516.2010, also known as HLA Evolved [15, 16, 17], has defined a Web Service (WS) interface, which makes it possible to run distributed simulations over the internet with web technologies.

An older technology for distributed simulation is Distributed Interactive Simulation (DIS) [18]. It is based on a fixed data model, which makes it less flexible than HLA, but which ensure compatibility. Data is exchanged by multi-cast, so the simulation components should be on the same local network. DIS is still frequently used and is in many cases compatible with HLA through a gateway.

3.3 Intelligent agents

An *agent* is an autonomous entity that observes through sensors and acts upon its environment using actuators in order to meet its design objectives. To be called intelligent, an agent also has to be reactive, proactive and social; meaning it must be able to react to changes in the environment, pursue goals and be able to communicate with other agents [19].

A Multi-Agent System (MAS) contains a number of agents that interact with one another. Each agent will influence different parts of the environment, and the agents are linked by some kind of organizational relationship. The agents in a MAS can be the same (homogeneous MAS) or different (heterogeneous MAS), and they can be cooperative or self-interested. Motivations for using a MAS can be to solve problems that are too large for a centralized agent alone, to allow interconnection and interoperation of multiple legacy systems, or to offer conceptual clarity and simplicity of design.

3.4 Context-based reasoning

CxBR is a paradigm for modelling the behaviour of intelligent agents. The motivation behind CxBR is the realization that people only use a fraction of their knowledge at any given time. The idea is to divide the knowledge into contexts in order to limit the number of possibilities for the action selection process. The following gives a short introduction to CxBR including some essential concepts. A more extensive description can be found in [4].

The contexts are organized in a context hierarchy consisting of a mission context, major and

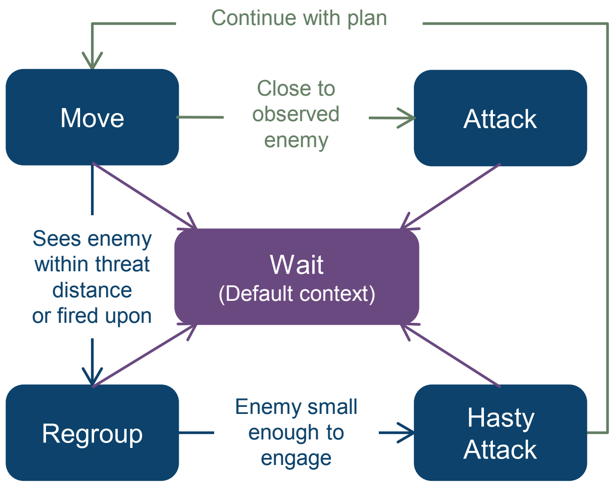


Figure 3.1   A context map defines all possible contexts and the transitions between them.

minor contexts. The *mission context* is a purely descriptive context, meaning it does not describe behaviour. A mission context contains a goal and a plan for reaching it together with parameters like objective areas, phase-lines, routes, etc., and a context map. A context map defines all possible transitions between the major contexts, as illustrated in figure 3.1.

Major contexts constitute the next level in the context hierarchy and are the ones controlling the agent. There is only one major context in control of the agent at any time, called the active context. A major context basically contains three kinds of knowledge: *action knowledge*, *transition knowledge* and *declarative knowledge*.

Action knowledge is knowledge about how the agent should behave in this context. Since our agents are battle command agents, the actions are commands to the subordinates, reports to the superior and possibly reports and/or requests to other agents at the same level. If a part of the behaviour is shared with other major contexts, this behaviour should be expressed as a *minor context*, which controls the agent for a short period of time. There can be unlimited levels of minor contexts, but one or zero should be sufficient. Minor contexts are not used in the example model presented in this report.

Knowledge of when to switch into another context is collected in the transition knowledge. This includes recognition of a situation leading to deactivation of the active context and activation of a better suited context. This knowledge can be contained in transition rules, with criteria for when the agent makes the transitions defined in the context map. The transition rules consist of both general, doctrinal reactions and scenario specific, planned transitions, and should include transition to a default context when no other context is applicable.

Declarative knowledge includes other properties of the context, e.g. parameters and a list of possible minor contexts.

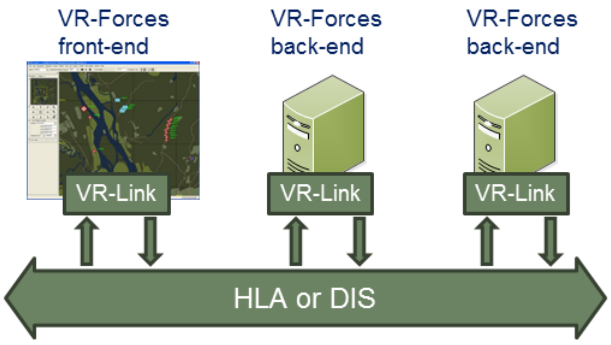


Figure 3.2 A VR-Forces simulation environment with two back-ends controlled by one front-end. In such a setup one back-end may simulate the blue force, while the other simulate the red force.

3.5 VT MÄK VR-Forces and VR-Link

VR-Forces is a simulation environment and framework for CGF applications [20]. It comes with a range of battlefield entities and weapon systems, but it is expected that the users extend and configure it according to their own requirements [21, 22, 23].

VR-Forces consists of a front-end application and a back-end application. The back-end is the actual CGF simulation engine, while the front-end is a graphical interface allowing a user to create, manage and play scenarios. A VR-Forces CGF scenario can be scaled up by running multiple front-ends and/or back-ends. All the VR-Forces applications communicate over a networking toolkit named VR-Link. VR-Link provides a unified and extensible application programming interface (API) that allows communication over multiple versions of both DIS and HLA. If the user needs to communicate over HLA using an extension of the RPR FOM or a totally different FOM, the VR-Link Code Generator application can be used to lighten the work. Figure 3.2 illustrates a typical VR-Forces simulation environment scaled up with two back-ends controlled by one front-end.

In addition to being highly configurable, both the VR-Forces front-end and back-end can be extended either by being embedded into another application or through plug-ins. In either case the extensions will be done using the C++ API provided for VR-Forces and VR-Link. It is also possible to control one or more back-ends through the VR-Forces Remote Control API. The Remote Control API consists of a set of C++ classes that can be used to make your own front-end or other type of remote controlling application.

Internally, the VR-Forces front-end utilizes the VR-Forces Remote Control API to perform scenario management, tasking, and all other remote control of the back-end. The Remote Control API creates messages in a closed VR-Forces proprietary format, wraps these messages inside `RawBinaryRadioSignal`-interactions and sends these interactions over HLA or DIS.

The VR-Forces back-end simulates forces as either individual entities or aggregated entities. An individual entity (normally referenced simply as an “entity”) represents e.g. a tank, an aircraft or a soldier, while an aggregated entity is a collection of entities typically corresponding to a higher echelon organizational unit (e.g. a platoon). Aggregated entities can be simulated in either disaggregated state or in aggregated state. In disaggregated state the subordinates are simulated as individual entities, but coordinated by an aggregate controller. In aggregated state the subordinates of the aggregate are not simulated separately. It is also possible to have aggregates of aggregates, i.e. you may have a company aggregate controller that coordinates several platoons, which again are coordinated by platoon aggregate controllers.

3.5.1 Entity models and behaviour

An entity model consists of sensors, controllers and actuators. Sensors detect different kinds of input from the simulated environment. VR-Forces provides several generic sensors, like visual sensor, infrared sensor, and active and passive radars. Each sensor has a parameter set describing the sensor, e.g. the maximum range. Controllers calculate what to do based on sensory input and models of the entity, weapon system, environment, etc., and send the result to the actuators. The actuators are what make the entity actually do something that affects the situation, e.g make the vehicle move, send a message or shoot.

The behaviours of the entities in a simulation are controlled through parameter values and tasks. Set-commands are used to set parameters of the entity, for example ordered heading, ordered speed, and rules of engagement. Task-commands are used to give simple tasks to the entity, like movement along a provided route, attack, and wait. An entity tasked to move along a route will follow the straight lines between the waypoints in the route, and not do any path planning. Attack means to move towards a given position while shooting at all detected enemies within fire range. The wait task is often used when the entity has no other tasks. The entity simply stays where it is, without doing anything, except from monitoring the environment with its sensors. It is possible to combine set-commands and task-commands in a plan using logic statements, like if, when, and while. Lua scripts can be used to model more complex behaviour [24].

3.5.2 Terrain

The terrain is represented in MÄK Terrain Format (GDB), which contains polygonal data with corresponding surface information and vector data with associated information. The polygonal data may consist of triangles, convex planar polygons, or a mix of these. The vector data consists of points and edges. The edges can be connected in closed shapes to form areal features, or they may form linear features, like roads. The associated information of a road may contain road width, its name, or other useful specifications. There are several categories of terrain types, for example water features include ocean, lakes, ponds, and shallow streams. Some of these categories are represented as properties of polygonal data, while others are associated with vector data. Whether or not a water feature is crossable depends on its properties, which are configurable.

Forests can be modelled as areal features or by single tree objects. Single trees are considered obstacles and cannot be moved through. Whether a forest can be moved through depends on the parameters associated to the areal feature. It is possible to allow movement through forests, or to mark the forest inaccessible.

Line of sight calculations are done to check whether a visual sensor detects other entities, and whether other entities can be shot directly at. The calculations are done by searching through the triangles in the terrain to check whether any triangles intersect the straight line between two given positions, for example two entities. When checking line of sight from an entity, it is possible to specify the height of the observer on the entity. Forests are normally not accounted for when determining line of sight. It is possible to specify that forests are totally opaque, but not that visibility gradually decreases through forests. Single trees can be modelled, and each tree will then be opaque. Inclusion of several single trees slows down the line of sight calculations.

3.5.3 Radio communication, perceived truth, blue force tracking

VR-Forces has a radio messaging system that is used for transmission of radio messages between entities. Each entity can be equipped with one or more radios and can send messages with any of its radios. The messaging system allows a message to be sent to one or multiple recipients with the same type of radio. As default VR-Forces does not model radio propagation and signal loss, but this can be added through either third-party software or self-developed models.

Spot reports represent one type of radio messages in VR-Forces. When spot reports are activated and a sensor of an entity registers another entity, it immediately sends a spot report to the rest of its force about the observation. There are five levels of entity identification. At the lowest level the entity is not detected. At the next level it is registered that there is an entity of unknown type and force, but its platform type is registered, for example ground. At the highest level all visible information about the entity is known, including type of entity and which force it belongs to. For each detection of an entity or update of the identification level, a new spot report is sent.

In addition to the ground truth containing all entity positions, each entity has a detection table containing the entities it has detected or received spot reports about. Since it is possible to disable broadcast of spot reports from all or a selection of entities, the detection tables of the entities in a force need not be identical.

The standard settings use ground truth for the entities in own force, but it is possible to use spot reports also for detection of own forces instead of ground truth.

3.5.4 Exercise clock

The exercise clock in VR-Forces manages the progression of time in the simulation. This is done by ticking all the individual simulation components. A component tick provides the current simulation time and executes the logic of the called component. The processing time needed to tick all the components of the simulation differ depending on which events occur. E.g. if an entity meets an obstacle, its actuator component might need to recalculate the route to the target location. Because of the differences in tick processing time, VR-Forces provides three exercise clock modes:

Variable-Frame Run-To-Complete mode advances the simulation time after each tick with the processing time of the completed tick. As a result the simulation time since the last tick, also referred to as a frame, depends on the processing time of the last tick. Because the processing time is affected by other processes running on the same computer, the simulation will not be repeatable. The frame length also depends on the amount and types of events that occur in the simulation. Long frames cause loss of time granularity, as the simulated time between component ticks increases. This becomes an even larger problem if the simulation is run faster than real-time, because the simulation time then is advanced with the processing time of the last tick multiplied with an acceleration factor. This mode should not be used in time-managed HLA federations.

Fixed-Frame Best-Effort mode advances simulation time by a fixed amount each frame, meaning the simulated time between two ticks will always be the same, even if the simulation takes more than the fixed amount of time to compute. If less than the fixed amount of time is used, the simulation sleeps until this amount of time has elapsed before continuing. When running the simulation faster or slower than real-time in this mode, VR-forces adjusts the fixed amount of time the simulation time should be advanced with for each tick. Instead of this behaviour, we want the sleep time to decrease when we try to run the simulation faster, assuming that the simulation usually are sleeping between ticks. The modification is described in section 6.1.3. This mode requires that the hardware can handle the frame-rate, and is then suitable for distributed use only in time-managed HLA federations.

Fixed-Frame Run-To-Complete mode advances simulation time by a fixed amount each frame, even if a frame takes longer than the fixed amount to compute, like in Fixed-Frame Best-Effort mode. The difference is that this mode does not sleep if the simulation takes less than the fixed frame time to compute. The effect is that the simulated clock will go faster or slower depending on processing time. In other words, the simulation will run as fast as possible with the given frame length. This mode is most useful for situations where only the end result of the simulation is interesting, while it is unnecessary to observe the simulation execution. Fixed-Frame Run-To-Complete mode is therefore not suited for interactive use and is suitable for distributed use only in time-managed HLA federations.

3.5.5 B-HAVE

B-HAVE [25] is a plug-in for VR-Forces that contains extended models for the behaviour of land based entities, i.e. lifeforms and ground platforms. The functionality is based on Kynapse [26], a library for modelling behaviour used in several video games. B-HAVE includes 3D path finding, complex behaviour like hiding, fleeing from an enemy and wandering. B-HAVE makes it easier to control entities as one does not have to specify routes to make sure entities avoid obstacles, move around lakes, etc. The entities can be told to go to a location and through B-HAVE figure out how to get there.

The path finding is based on construction of a navigation mesh and a corresponding graph. These are constructed from the terrain model in a path data generation process. The path data depend on the type of entity they are created for. The desired path is found by an A* search through the graph. B-HAVE includes dynamic smoothing of the path during path following.

The path planning in B-HAVE has limitations. It distinguishes only between the terrain types roads and non-roads, and it takes inaccessible areas into account. The path planning therefore produces a route that avoids obstacles and is as short as possible. This path planning works best along roads and in urban areas and is not particularly suited for rural areas, since slope and soil types are not considered.

4 System design

We have developed a demonstrator of a simulation system capable of autonomous simulation of battalion operations. This simulation system is integrated with a C2IS so that the C2IS can be used to visually view and create orders, in addition to presenting ground truth and perceived truth from the simulation system. The simulation system is capable of receiving and executing orders created by the C2IS and providing reports back to the C2IS. The C2IS is also used to define the ORder of BATtle (ORBAT) and to provide the initial positions of friendly units. C-BML and MSDL documents are used to exchange orders, reports and scenario definitions between the C2IS and the simulation system.

In order to simulate battalion level operations autonomously, the simulation system must be able to understand and plan higher level commands like "seize area x" or "support by fire unit y", and be able to react to unplanned events according to doctrine. In commercially available CGF systems, like VR-Forces, only lower level tasks like "move along route" and "move into formation" are supported. Since VR-Forces can be extended by writing plug-ins, it could be possible to add models of higher level tactical behaviour and doctrinal knowledge in VR-Forces. However, we chose to build a MAS independent of the CGF system in order to make it easy to replace VR-Forces with another CGF system in the future. This MAS includes tactical and doctrinal knowledge and is used to control the entities in VR-Forces.

The basic system architecture is illustrated in figure 4.1. Next we will give a brief description of the functions and roles of the C2IS, the MAS and the CGF system, and provide details about the communication between them. The MAS will be explained more thoroughly in section 5, and details about extensions and configurations in the CGF system VR-Forces will be described in section 6.

4.1 The command and control information system

The C2IS utilized in our system configuration is NORTaC-C2IS. Through previous NATO science and technology groups this C2IS has been extended with functionality for defining and presenting the information necessary for creating C-BML orders. This extension was implemented by

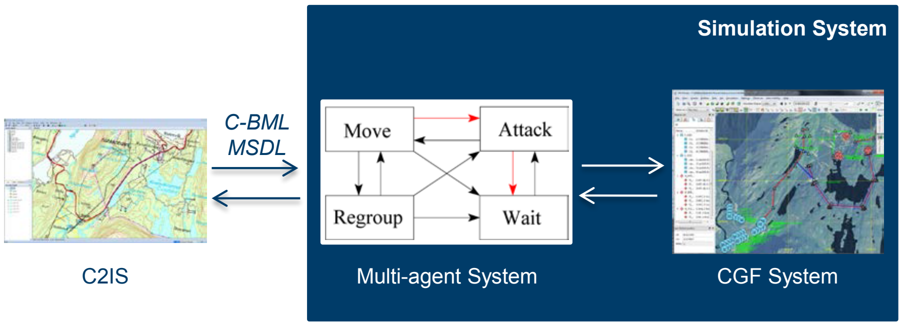


Figure 4.1 The MAS gets as input an operational order at the battalion level and produces commands to the CGF system. The agents' actions are influenced by reports from the CGF system. Reports are sent back to the C2IS.

KDS in collaboration with FFI. Orders created through this extension is stored in a NORTaC-C2IS database that conforms well to the Multilateral Interoperability Programme (MIP) Joint Consultation, Command and Control Information Exchange Data Model (JC3IEDM) [27].

In order to extract order definitions from the NORTaC-C2IS database and convert them into C-BML orders, the FFI C2-gateway was created. This C2-gateway use SQL to extract order data from NORTaC-C2IS and converts this data into C-BML documents that is sent through a message broker. The C2-gateway is also capable of receiving C-BML reports over the same message broker and inserting them into the NORTaC-C2IS database through SQL. The C2-gateway uses SQL to extract the ORBAT and positions of units defined in the ORBAT from the NORTaC-C2IS database. The ORBAT and the initial unit positions are utilized to construct an MSDL document for scenario initialization. Like with the C-BML documents, this MSDL document is also sent through the message broker to the simulation system.

NORTaC-C2IS combined with FFI C2-gateway thus function together as a C-BML capable C2IS. More details about the NORTaC-C2IS extension and the FFI C2-gateway can be found in [9, 28, 29].

4.2 Communication between the C2IS and the MAS

There currently exist several experimental infrastructures for exchanging C-BML and MSDL documents. Through the C-BML NATO science and technology groups FFI have participated in multiple experiments and demonstrations using different versions of the Scripted BML (SBML) Server developed by George Mason University (GMU) [30, 31].

The SBML Server utilizes a GMU-developed language to define mappings from XML documents (MSDL documents and different types of C-BML documents) to a JC3IEDM compliant database. This mapping functionality is used to persist C-BML and MSDL documents. Documents are inserted into the server through either a SOAP or a REST web service. Documents sent to the server is published to clients that has connected and subscribed using a Message Oriented Middleware

(MOM) called Java Message Service (JMS). The server also has functionality for merging multiple MSDL documents. The FFI C2-gateway is capable of utilizing the functionality provided by the SBML Server.

FFI has also updated the FFI C2-gateway to support the Coalition Battle Management Services (CBMS) created by Virginia Modeling, Analysis and Simulation Center (VMASC), but it has never been used or tested. Like the SBML Server, CBMS is also capable of persisting and publishing XML documents. CBMS uses REST-based web services for inserting documents and Web Socket connections for allowing clients to subscribe to documents.

When we created our system configuration, both the SBML Server and CBMS were considered as message brokers. As already mentioned the FFI C2-gateway has support for both. When we created the system configuration described in this report, we found we did not need an infrastructure with persistence, filtered subscription capabilities or MSDL merging. Instead we found that we needed only a simple message broker that ensured that messages sent by one client were received by all other clients. We also wanted the message broker to be maintenance free (e.g. not needing to use external tools to reset a SQL database) and really simple to configure.

Both the SBML Server and CBMS fall short on these two last requirements. The SBML Server has evolved to become relatively complex to install and configure, in addition to requiring clients to use multiple communication technologies (i.e. REST or SOAP for inserting documents and JMS for filtered subscriptions). CBMS appears simpler to install and configure, but also requires clients to utilize two different communication technologies.

To meet our requirements for a lightweight and simple solution we chose to develop a basic message broker using Web Sockets. This message broker consists of a Web Socket server that each client connects to. The only functionality of the server is to forward received messages (i.e. MSDL or C-BML documents) from a sending client to all other connected clients. The message broker server requires no configuration or maintenance. The only configuration needed is for the clients to know the port and host address of the server.

4.3 The multi-agent system

A battalion order consists of company tasks and how these should be synchronized. The MAS decomposes the company tasks to low-level CGF commands through a hierarchy of intelligent agents. There is one agent for the battalion, one for each company in the battalion and one for each platoon. The agents represent the leaders with staff of these military units, and we try to imitate the planning and decision making. Using a MAS for this purpose makes the design clear and simple, and it becomes easy to understand for military experts. Simulation of the real chain of command also prepares the system to be used for other tasks, like studies of communication in the organization hierarchy.

How the agents carry out the ordered tasks depend on the terrain and observed enemies, which are information provided by the CGF system. The agents react according to doctrine to unplanned events, like the CGF entities being fired upon while not in an attack.

4.4 The CGF system

Units below platoons (i.e. squads, vehicles, soldiers) are not represented in the MAS and the decomposition of platoon tasks to tasks for individual entities is handled by the CGF system. The platoon aggregates are simulated in disaggregated state in VR-Forces. This was done foremost because it allows a better visualization of close-combat situations.

4.5 Communication between the MAS and the CGF system

In order to allow the platoon agents to control the platoon aggregates in VR-Forces, simulation state, reports and commands had to be exchanged between the MAS and VR-Forces. As described in section 3.5, VR-Forces supports the communication standards HLA and DIS. We chose HLA as the communication standard for all our simulation system internal communication, because it is more flexible and extensible than the older standard DIS. As VR-Forces utilizes the RPR FOM, we also chose to use this FOM [13].

4.5.1 Low-level BML

In addition to the simulation state already expressed through the RPR FOM, the MAS must be able to send commands and scenario management messages to VR-Forces and receive reports back. The VR-Forces Remote Control API has the necessary functionality to support these requirements, but would bind our MAS to VR-Forces. Instead we developed Low-level BML [3]. This language was developed in collaboration with TNO and consists of three main parts:

1. commands used by the MAS to instruct CGF entities,
2. reports from the CGF entities related to task status and status of the tactical environment used by the agents to perform higher-level reasoning and
3. scenario management functions used by the MAS to initialize the CGF system and to create control features in the CGF.

As we use HLA and the RPR FOM, we started out with making an extension of the RPR FOM that defined interactions for the Low-level BML constructs. We later decided to instead encode the Low-level BML constructs using Google Protocol Buffers [32] and wrap these encoded messages inside the RPR FOM “Application Specific Radio Signal” interaction.

4.5.2 Time management

The battle command simulation done by the MAS and the entity simulation done by VR-Forces must be synchronized in order to collaborate. I.e. tactical decisions must be made in a timely fashion relative to state and situation changes, and the entities should not run ahead of the tactical decisions.

A simulation can be run in either real-time (time progress at the same rate as your watch) or scaled-time (time progress faster or slower than real-time). In simulation jargon, there is “wall clock time” and “simulated time”. Wall clock time follows the human perceived real time, while the simulated time is the time as perceived by the simulation. In a real-time simulation, the simulated time rate of advance corresponds to the wall clock time. But in a scaled-time simulation they do not correspond.

As our system configuration should support operational planning, the simulation system must be able to run faster than real time. It must also be possible to pause and resume the simulation without the different components getting out of synchronization.

The HLA time management functionality could have been used to synchronize the MAS and VR-Forces [33]. This would have allowed us to ensure that none of the federates could run ahead of others. HLA time management was not utilized for this version because in the beginning of the project we got the impression that VR-Forces did not support this HLA feature. We later found that the VR-Forces back-end actually supports time management. Another factor was lack of time and that time management was not yet fully supported in FFI HLA Java library, HlaLib [34].

Instead of using HLA time management we chose to add a tick-message to Low-level BML, and made the VR-Forces back-end send this tick-message with the current simulation time approximately once every simulated second. In the MAS this tick-message triggered processing. As there is no time management, VR-Forces will not wait for the MAS to provide commands as a result of the current tick. A possible problem with this approach is that the MAS may not be able to process incoming events fast enough and therefore may lose data or start to lag behind the VR-Forces simulation. This solution was selected because it was quick and easy to implement and because the mentioned issues only manifested themselves if VR-Forces was run at maximum speed.

4.6 Final system configuration

Our final system architecture is illustrated in Figure 4.2. This figure illustrates all the applications the total system consisted of and how these applications communicated.

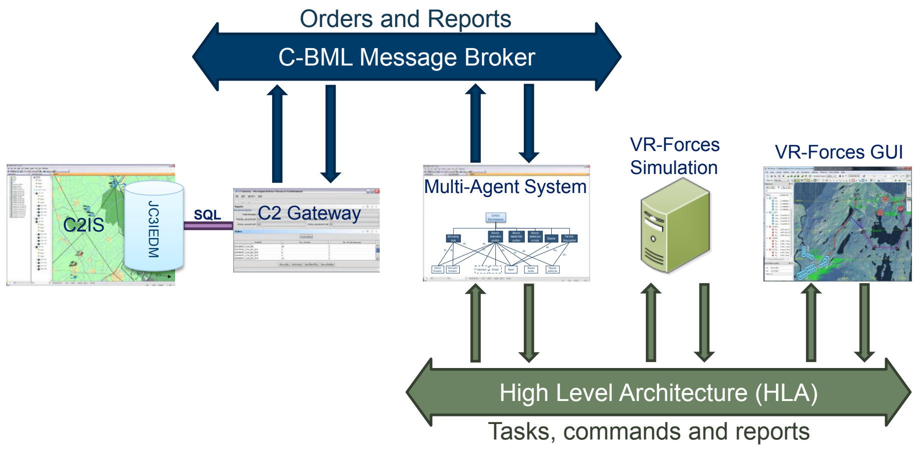


Figure 4.2 The MAS receives as input a C-BML order at battalion level and produces commands to the CGF system. The actions are influenced by reports from the CGF system. Reports are sent back to the C2IS.

5 Multi-agent system framework

Having described the total system configuration, we will now focus on the design decisions and implementation of a framework for the MAS. The first subsection describes the requirements for the framework. Section 5.2 describes the approaches investigated, while section 5.3 describes the architecture of the chosen approach. Implementation details, including how the different parts of the framework work together, are described in section 5.4.

5.1 Requirements

The MAS framework shall allow intelligent agents to be implemented using CxBR concepts. The agents shall act based on C-BML formatted battalion orders received from a C2IS. In order to perform reasoning the behaviour models shall have access to data about the simulated environment and react to events in the simulated environment, where the simulated environment consists of the entity state simulated by the CGF system and control measures defined in the received C-BML orders.

Entity state shall be accessed through HLA interactions and objects modelled using the RPR FOM. The agents shall interact with the entities simulated in the CGF system through Low-level BML commands sent over HLA. Entity events of interest for the agents shall be communicated as Low-level BML reports sent by the CGF system.

A MSDL document from the C2IS representing the ORBAT shall be used to instantiate a battalion agent, company agents and platoon agents. As the platoon agents shall correspond to platoon aggregates in the CGF system, the MAS shall trigger the instantiation of these platoon aggregates through Low-level BML scenario management messages.

5.2 Approaches investigated

This section provides an outline of implementation approaches we discarded and a short discussion on why we chose the current approach.

In the beginning of the MAS framework development process, we started out researching existing agent frameworks and rule-engines. We did this in order to find a way to represent the concepts needed to implement intelligent agents and CxBR.

Early in this process our collaborating partners at TNO suggested to use the Java-based middleware “Pogamut” to implement our agents. Pogamut is a software library for developing video game agents [35]. While there are similarities, a CGF system like VR-Forces is not the same as a video game. These differences aside, the concepts and architecture found in Pogamut did not appear to fit well with our chosen CGF system and our research plans.

Both TNO and FFI therefore chose to abandon Pogamut and instead we started investigating the Jadex agent framework [36]. Jadex allows for programming agents in XML and Java, using Belief-Desire-Intetion (BDI) reasoning concepts. Our biggest issue with this framework was the binding to BDI-based reasoning. The paper [37] describes how CxBR concepts maps to BDI concepts. We considered utilizing these mapping ideas in order to realize CxBR concepts in Jadex. After some considerations we found Jadex to be unsuitable for modelling CxBR concepts and we considered the Jadex architecture to put too many restrictions on how our concepts could be realized. Therefore we chose to look for a more generic approach that we could tailor to our needs.

CxBR concepts like switching between contexts, firing actions and so on can be modelled and implemented in many ways. For example some implementations utilize neural networks or rule engines (often referred to as “expert systems”). Neural networks can (among other things) be used to implement self-learning agents and agents imitating how the human brain actually works. Both are complex areas we found too complex for our current framework.

As we wanted to represent the CxBR action knowledge and transition knowledge as rules, we researched existing Java-based rule-engines. The most mature and most used Java rule-engines appears to be JBoss Drools [38], IBM ILOG JRules [39] and Jess [40]. The latter is a Java-port of the NASA-based CLIPS rule-engine [41]. All three appears to use dedicated rule languages to express rules and facts.

When choosing which rule-engine to use, we wanted an engine that could handle temporal reasoning (i.e. event processing) and that integrated well with complex calculations implemented in Java. JRules and Drools both have dedicated support for event processing, while it appears you have to emulate it yourself in Jess. We wanted to have some complex calculations implemented in Java because we found it cumbersome or impossible to implement these calculations in the dedicated rule languages used by the rule engines. We therefore chose to investigate Drools further, as it is open source and allow easy integration between rules and logic written in Java. JRules was abandoned as it is closed source, expensive and very little information was available

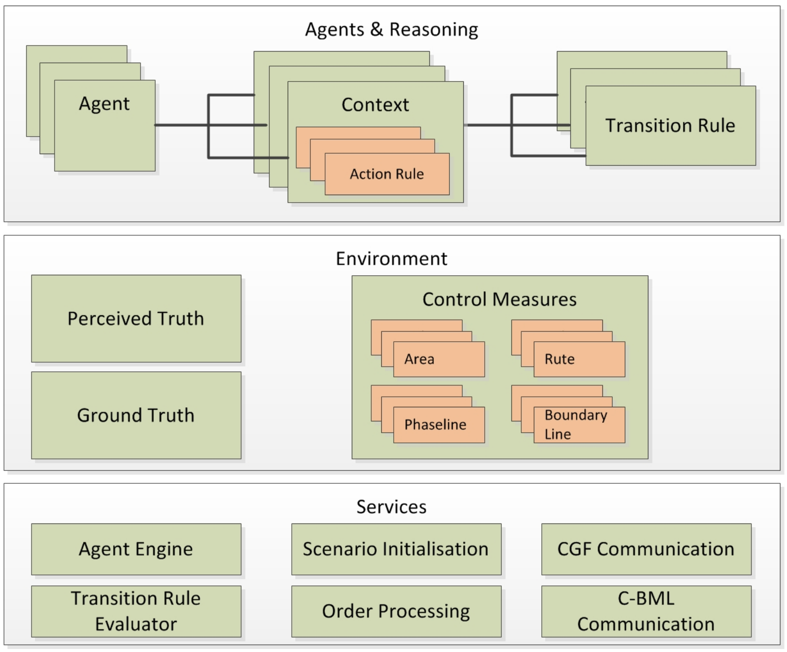


Figure 5.1 Illustration of the parts and components of the MAS framework, roughly separated into Agents and Reasoning, Simulation Environment and Services.

for evaluating it. Jess was abandoned because it used a cumbersome LISP-format and appears not to offer the integration we wanted with calculations written in Java.

Through investigating Drools we found that this rule engine, like the other rule engines we considered, expects all facts to be inserted into a knowledge base before finding and firing rules that match the current set of facts. This became an issue because we only wanted to fire our complex Java calculations depending on the situation. E.g. in some circumstances we only wanted to calculate the force ratio against the enemy upon being fired upon. We could have performed all the possible complex calculations once for each and every event and inserted the result into the knowledge base, but the computational overhead would have been too large. Also, while Drools offer good integration between rules and Java logic, we found it cumbersome and too restrictive.

After having fought with Drools some months, we decided to abandon dedicated rule engines altogether and instead create our own dedicated CxBR-based MAS framework in plain Java. This framework allowed us to conceptualize and implement the exact concepts we needed to fulfil our requirements, instead of adjusting our concepts to a pre-exising rule engine.

5.3 Architecture and design

The MAS framework has been realized as a set of components and services designed to allow simulation of tactical decisions through intelligent agents modelled using CxBR. The different parts of the architecture is described below and illustrated in figure 5.1.

Agents and CxBR are a set of classes used to represent and instantiate agents, contexts and the other CxBR concepts.

Simulation Environment is designed as components for allowing the agents and reasoning functionality to operate on simulation state and control measures.

Services include communications services that provide services for communicating with the C2IS and with the CGF system, and internal services that are necessary to drive the internal functionality of the framework. The internal services are for example services for scenario initialization, order processing, transition rule evaluation and the agent engine itself.

The MAS framework is designed to be event-driven. There are three types of events that trigger processing: MSDL documents, C-BML order documents and tick messages. MSDL documents and C-BML order documents are received from the C2IS, while tick messages are received as Low-level BML messages from the CGF system. In the current system implementation the MSDL document contains the ORBAT and the initial positions and orientations for the platoons.

Reception of a MSDL document triggers initialization of the MAS. Before receiving the MSDL document, no agents are instantiated in the MAS and no friendly entities or aggregates exists in the CGF system. In other words; when processing a MSDL document, the MAS framework creates agents for the battalion, companies and platoons, in addition to sending commands to the CGF system for creation of entities and aggregates for the platoons. Because the aggregate creation commands are provided with the initial positions and orientations defined in the MSDL document, the CGF system can instantiate the platoon aggregates and their subordinates at the correct locations. The actual implementation of this scenario initialization process is described in section 5.4.2.

The lowest-level agents, i.e. the platoon agents, command CGF aggregates and receive state from the CGF aggregates. We wanted it to be transparent to the agent behaviour if it should delegate to agent subordinates or to a CGF aggregate. We therefore created the concept of a “CGF agent”, which is a special type of agent that takes care of receiving state and reports from the platoon aggregates over HLA and sending commands the other way. During the scenario initialization process all the lowest-level agents are assigned a CGF agent as a subordinate. The creation and initialization of these CGF agents are further described in section 5.4.2. The structure and functionality of the CGF agents are explained in section 5.4.3.

After being initialized, the MAS is ready for receiving a battalion order. Upon receiving a C-BML order document the MAS framework will extract the company tasks in the battalion order and convert them into company missions and mission start rules. These company missions and start rules will be assigned to the battalion agent. The services and concepts used to realize the order processing is further described in section 5.4.4.

When the MAS is initialized and a C-BML order has been received and processed, the agent behaviour must be executed. This execution is triggered by tick messages. With our current configuration the CGF system produces one tick message per simulated second. I.e. if the CGF system is run faster than real-time, the wall-clock time between each tick message will be less than one second.

Upon receiving a tick message, the MAS framework first updates all the agents before triggering the actual agent behaviour processing. Updating an agent means different things if it is a CGF agent or a battle command agent, but both has in common that they process new commands from the superior. In addition the CGF agents copy the state from the HLA object representing the corresponding CGF aggregate, checks if the aggregate has completed the last assigned task and evaluates if it has threatening enemies. See section 5.4.3 for further details about the CGF agent functionality.

In addition to processing new commands, the battle command agents also trigger evaluation of their CxBR transition rules when being updated. This ensures that the agents utilize the correct CxBR contexts when the agent behaviours are triggered as the next step in the tick message processing.

After all agents have been updated, the agent behaviour is triggered for each agent. For all agents the behaviour execution starts with checking if the current mission is completed. If it is completed, a mission-completed report is sent to the superior. Further, agent behaviour execution means ticking the current CxBR context for battle command agents, and sending of Low-level BML commands to the CGF system for the CGF agents.

An important design decision is that the tick message process first updates all agents from lowest level and up, before triggering agent behaviour for all agents from lowest level and up. I.e. the CGF agents are updated before the platoons, the platoons before the companies, etc., before doing the same sequence for the agent behaviour. This was done because all decisions and behaviour of an agent is based on the reports from and states of its subordinates. E.g. to compute the position of a company agent, the positions of its subordinate platoon agents are needed and so on. Also, updating all agents before triggering the agent behaviour, ensures that all agents base their decisions on the same state (otherwise two agents might see the state of a third agent differently depending on the order they are processed). Agent functionality and reasoning is further described in section 5.4.3.

5.4 Implementation

This section describes the implementation of the different concepts and components described in the previous section. We will not describe all classes or services, only enough to explain how the framework works and make it easy to understand the code.

5.4.1 Communication services

The MAS framework has two communication interfaces: one for the exchange of C-BML and MSDL documents with the C2IS and one for low-level commands, reports and scenario management messages with the CGF system. The following sections describe how these two are implemented as communication services.

Communication with the C2IS is implemented in the `C2Communicator` service. As described in section 4.2, this service is a WebSocket client that receives C-BML and MSDL documents and sends C-BML status report documents.

The MAS framework communicates with the CGF system over HLA. HLA communication is controlled by the service `CgfCommunicator`. This service utilizes the FFI-developed library *HlaLib* to manage all interaction with HLA [34]. In addition to forwarding Low-level BML messages created by the other services, the `CgfCommunicator` service provides an API to other parts of the system for creating entities, aggregates and control measures (routes, areas and phase-lines) in the CGF system. The create-methods construct the corresponding Low-level BML messages. All Low-level BML messages are sent to the CGF system by first being serialized to a byte array using Google Protocol Buffers, before being wrapped in a RPR FOM *Application Specific Radio Signal*.

The `CgfCommunicator` service also listens for ground truth information about aggregates published by the CGF system, and publishes this information directly to the C2Communicator, which generates and sends C-BML position reports. These aggregates may be both aggregates defined in the VR-Forces scenario and aggregates created by the MAS framework.

The service `BmlMessageListener` is registered as a listener for incoming RPR FOM *Application Specific Radio Signals*. When such a radio signal is received, the contained data is decoded into a Low-level BML report using Google Protocol Buffer and processed according to report type. The supported report types are *TickReports*, *TaskReports* and *SpotReports*. Tick reports are forwarded to the `AgentEngine`, task reports are sent to the corresponding CGF agent, while spot reports are processed by the `PerceivedTruth` service.

5.4.2 Scenario initialization

Upon receiving a MSDL document, the `C2Communicator` calls the `ScenarioInitializer` service to process the document. The `ScenarioInitializer` extracts all unit descriptions from the MSDL document and builds up an internal "unit description" tree. After this extraction process, the `ScenarioInitializer` utilizes the `CgfCommunicator` service to create aggregates in the CGF system for each unit that does not have subordinates defined in the MSDL document. The created aggregates are initialized with name, location, type, heading, formation and a default rules of engagement set to hold-fire.

After creating the CGF aggregates, the `ScenarioInitializer` pushes the unit description tree to the `OrderOfBattle` service. This service takes care of instantiating and initializing the battle command agents. The agent initialization process uses the agent type (based on symbol identifiers from the MSDL document) to fetch a "agent description" from the `DescriptionManager` service. This agent description is further utilized to configure the

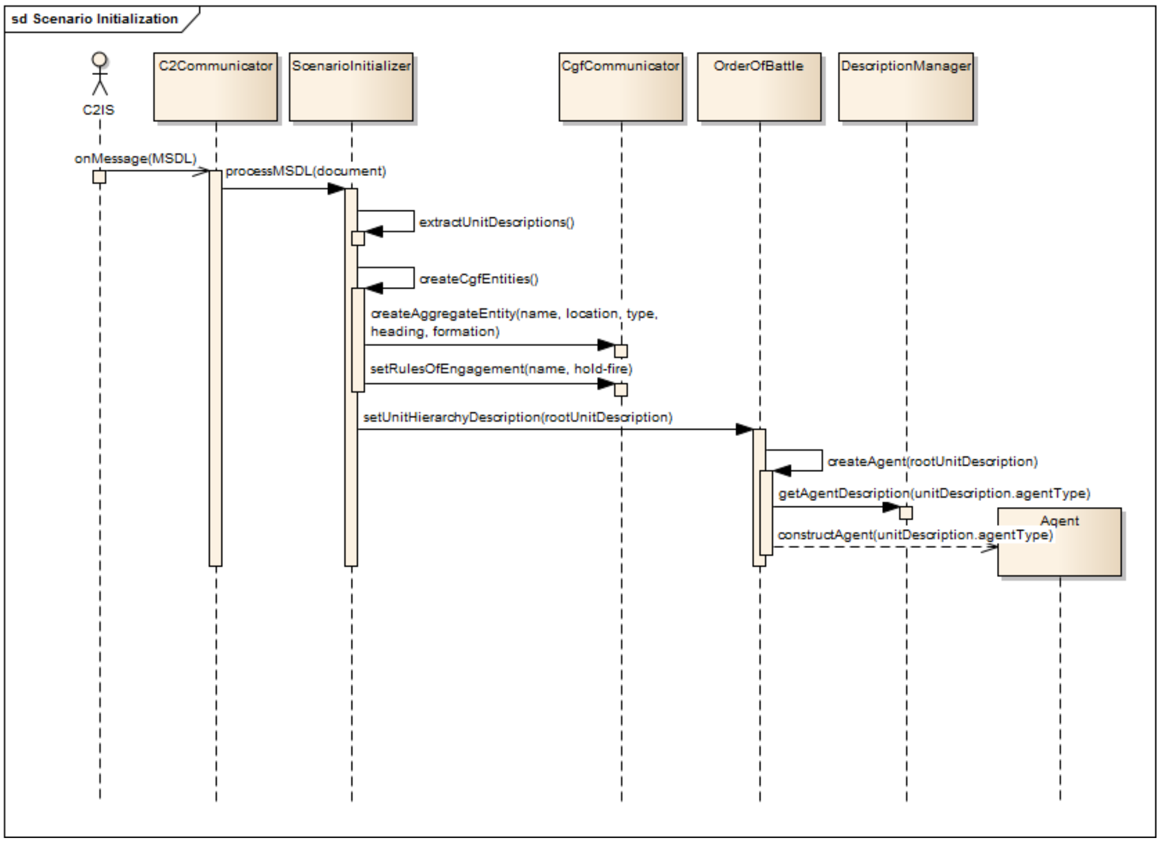

sd Scenario Initialization
C2IS
C2Communicator
ScenarioInitializer
CgfCommunicator
OrderOfBattle
DescriptionManager
onMessage(MSDL)
processMSDL(document)
extractUnitDescriptions()
createCgfEntities()
createAggregateEntity(name, location, type, heading, formation)
setRulesOfEngagement(name, hold-fire)
setUnitHierarchyDescription(rootUnitDescription)
createAgent(rootUnitDescription)
getAgentDescription(unitDescription.agentType)
Agent
constructAgent(unitDescription.agentType)

Figure 5.2 Upon receiving a MSDL document from the C2IS, the scenario initialization process creates aggregates in the CGF system and a hierarchy of battle command agents in the MAS.

new battle command agent instance with transition rules and a default context. The scenario initilization process is illustrated in figure 5.2.

The `DescriptionManager` service provides "agent description" for each potential "agent type". An agent description consists of a set of contexts, a default context, transition rules for each context and universal transition rules. All this is used to correctly initialize a battle command agent. To make the framework easy to configure, the `DescriptionManager` service can e.g. extracts its descriptions from a XML file. For the current implementation we chose to use a coded description manager. I.e. the agent descriptions are defined in Java code.

5.4.3 Agents

All agents implement the interface `Agent`. This makes it transparent for the scenario execution algorithm (see 5.4.5) if it is a CGF agent, platoon agent, company agent or battalion agent that is being updated or ticked. The `Agent` interface also makes it possible to implement agent logic (e.g. CxBR rules) independent of actual agent type. The `Agent` interface is illustrated in listing 5.1.

A core concepts shared by all the agents is the *event log*. The event log is a kind of queue that

```java
public interface Agent {
    String getName();
    void setName(String name);
    Agent getSuperior();
    void setSuperior(Agent superior);
    List<Agent> getSubordinates();
    void addSubordinate(Agent agent);
    void setTransitionRuleEvaluator(TransitionRuleEvaluator
        transitionRuleEvaluator);
    Context getContext();
    void setContext(Context context);
    List<Context> getPlan();
    WorldLocation getLocation();
    EventLog getEventLog();
    void addEvent(Event event);
    void update(double timestamp);
    void tick(double timestamp);
    PerceivedTruth getPerceivedTruth();
}
```

Listing 5.1 The interface that all agents must implement.

is filled up with events between each tick from the CGF system. These events may be mission commands, action commands and reports. All agents can produce and send events to any other agent. In addition CGF agents can receive task completed reports from the CGF system (received as a Low-level BML report via the `BmlMessageListener` service).

The event log allows an agent to differ between new and old events, in addition to the sequence of events. The `EventLog` implementation provides a set methods and predicate classes for searching and filtering events.

All the agents also share a common `PerceivedTruth` service. This service receives Low-level BML spot reports via the `BmlMessageListener` and maintains a simple operational picture shared by all the agents. By "operational picture" we mean tracking of position and type of enemy and neutral entities based on spot reports from all friendly entities.

Further, the agent implementations are divided into two main groups: battle command agents and CGF agents.

5.4.3.1 Battle command agents

As the functionality of all battle command agents is mostly the same, they all extends from the class `AbstractAgent`. The battalion agent differs from the company and platoon agent in that it does not have a superior. Platoon agents have CGF agents as subordinates, but this is transparent because all agents implement the `Agent` interface.

The MAS framework allows battle command agent implementations to be specialized according to agent type. The core agent classes are `Platoon`, `Company` and `Battalion`, but these can be extended with e.g. `MechanizedPlatoon`. These extension classes can override methods in the `AbstractAgent` with more specialized functionality. E.g. when processing received mission commands, a specialized agent class can override the default `MissionPlanner`. The functionality of the `MissionPlanner` is further described in section 5.4.6.

It is worth noting that the `Battalion` class overrides the base functionality for processing new commands. This is because the battalion agent does not have a superior that commands it, but instead receives battalion orders from the C2IS and schedules the company missions defined in these. This process is further detailed in section 5.4.4.

It should also be noted that as default the location for a battle command agent is calculated to be in the center of all the subordinate locations.

5.4.3.2 CGF agents

As described in section 5.3, the CGF agents shields the battle command agents from HLA and makes it transparent that the platoon agents actually are simulated in the CGF system.

The `CgfAgent` class makes this possible by using HlaLib to copy state from the corresponding HLA object (simulated by the CGF system). It also converts *mission commands* (received in the event log) into one or more Low-level BML commands using the `CgfCommandProcessor` service and sends these commands to the CGF system via the `CgfCommunicator`.

5.4.4 Order processing

When the `C2Communicator` service receives a C-BML order it calls the `OrderProcessor` service to process the order. The `OrderProcessor` service extracts all company tasks and control measures with help from the `IbmlParser` service. The control measures are sent to the CGF system through the `CgfCommunicator` service. The company tasks are linked to the corresponding company agents and converted to CxBR missions. The list of company mission contexts together with rules for when they should be initialized are added as a mission command to the battalion agent's event log.

Note that the `OrderProcessor` service must be extended when we want the MAS to support more task verbs and control measures.

5.4.5 Simulation execution

The core service of the MAS framework is the `AgentEngine`. Every time the MAS framework receives a tick message from the CGF system, this service drives the MAS framework by first updating all the agents from lowest level and up (CGF agents - platoon agents - company agents - battalion agent) and then triggering the agents' behaviour.

5.4.5.1 Updating the CGF agent

Updating a CGF agent involves several steps:

1. *Update Event log:* The events which have been added to the agent's event log since last update is made available to the agent.
2. *Update State from CGF:* The agents state is updated by copying HLA-data from the entity object corresponding to the CGF agent.
3. *Report Threatening Enemies:* The agent sends reports about potentially threatening enemies to its superior by adding a report to its superior's eventlog.
4. *Process Mission Commands:* If the event log contains new mission commands, these are translated to Low-level BML by the `CommandProcessor` service. A mission command might be converted into more than one BML task and/or set request, e.g. the mission context *Move* generates the task sequence *MoveIntoFormation* and *MoveAlongRoute*. Such a sequence is referred to as the *plan*. The agent will abandon its current plan when it receives a new mission command.
5. *Check Task Completed:* If the agent's current task is completed, it is deleted from the plan. This corresponds to a major context being completed.

5.4.5.2 Updating the battle command agent

Updating a battle command agent (platoon, company or battalion) involves a few slightly different steps:

1. *Update Event log*
2. *Process Mission Commands:* The agent abandons its current plan and uses the service `MissionPlanner` to generate a plan for the new mission context. The new plan is a sequence of major contexts.
3. *Evaluate Transition Rules:* The service `TransitionRuleEvaluator` will evaluate all relevant transition rules, based on the current major context, and possibly change major context. If the agent just received a new mission context, the new current major context will be the first in the new plan.
4. *Check Context Completed:* If the current major context is completed, and it is not the last in the agent's plan, the major context is deleted from the plan.

5.4.5.3 Executing behaviour of CGF agent

After all the agents are updated, their behaviour is triggered. For the CGF agent that means sending a report to the superior if it has completed its current mission context and sending commands to the CGF system. All new set requests are sent at once, but only the first task in the plan is sent to the CGF system, provided that it has not been sent at an earlier time step.

5.4.5.4 Executing behaviour of battle command agent

The battle command agents will equivalently send a report to their superior if they have completed their mission and then tick its active major context. Ticking a major context involves five steps:

1. *Initialize:* Some actions are to be performed when an agent enters the context. This step is only carried out if the context has status *not initialized.*
2. *Evaluate Action Rules:* Perform actions based on reports and knowledge about the current situation. The current situation is available through the `PerceivedTruth` service.
3. *Evaluate Context State:* If all subordinates have completed their mission, the state of the major context is changed to *completed.*

The `PerceivedTruth` service is important for the agents' decision making. It contains a list of all enemies observed by friendly forces, and also all friendly entities observed by enemies. The last is only used to report back to the C2IS. Every time a CGF entity reports a spotted entity of another force, the observed information is merged with earlier observations. The service also provides a function for extracting all spotted enemies within a given distance of a location.

5.4.6 CxBR implementation

The CxBR paradigm was explained in section 3.4. The key concepts were *mission contexts*, which are planned as a sequence of *major contexts*. The *major contexts* contain *action rules* and *transition rules*, and an agent has always one *active context* that is in control of the agent's behaviour. All these concepts are represented in the MAS framework.

The agents receive mission contexts, or as we have called them, *mission commands*, from their superior. In the current implementation an agent does not store the mission command object explicitly throughout the mission, but forgets what the mission is as soon as it has made a plan. The idea is that all information needed to complete the mission lies in that plan. Note that the mission command will always be available from the agent's event log, as nothing is ever deleted from this log.

The different major contexts are implemented as objects independent of the agent type, meaning the same class is used to instantiate contexts for platoon and company agents. All action rules are implemented as functions inside the context classes, except for the start rules used by the battalion agent. Transition rules are also tied to major contexts, but these are implemented as separate classes.

Which transition rules that are available to which major contexts is configured in the `DescriptionManager` service. This service also makes it possible to define which contexts should be available for which agents, and add universal transition rules which are available in all contexts for that agent type. In the experiments we used the same configurations for all agents.

The `TransitionRuleEveluator` service uses the configurations from the `DescriptionManager` to evaluate the relevant transition rules.

When an agent receives a new mission command, it uses the `MissionPlanner` service to make a plan for that mission context. This `MissionPlanner` service is the same for platoon and company agents. The CGF agent has its own planner, and the battalion agent does not have a planner, since his plan always consists of only one and the same major context, which we called *Schedule Order*.

5.4.6.1 Planning

In [42] we propose a three step planning strategy. In the first step a basic plan for the current mission context is retrieved. In step two the basic plan is adapted to the current situation by pruning away irrelevant contexts and specifying the transition rules, and in step three the general contexts from the basic plan could be replaced by more specific versions. Step three was only described as a future possibility, since the current model does not contain specialized versions of the basic major contexts used in the basic plans.

To explain how planning is implemented in the framework, we will use the planning of mission context *Seize* as an example. In the current implementation the *Seize* mission requires a phase line and routes for all the platoons. The basic plan for mission context *Seize* is (*Move* → *Move Cautiously* →*Attack*). The phase line defines when to change from Move to Move Cautiously, and the transition from Move Cautiously to Attack happens at a fixed distance from observed enemies in the objective area.

Instead of adding transition rule (Cross phase line → *Move Cautiously*) to major context *Move* and (Distance x from observed enemy → Attack) to major context *Move Cautiously* we decided to make goals for each context in the plan and then use a general transition rule (Context completed AND plan not empty → Next context in plan). This general transition rule can safely be added to all major contexts, meaning it is independent of mission context. Transition rule (Cross phase line → *Move Cautiously*) could not be added safely to major context *Move* because other agents with other mission contexts could enter this context, and the transition rules are connected to the class and not just one instance of the class. A goal for a major context on the other hand, is specific for one instance of that class.

When planning mission context *Seize*, the goal for each major context is defined by splitting the route for the mission into three parts, one for the *Move* context, one for the *Move Cautiously* context and one for the *Attack* context. A context is completed when the agent is at the end of the route. The route for context *Move* is the part of the total route that is between the agent and the phase line. If the route starts after the agent has crossed the phase line, major context *Move* is pruned away. The part of the route belonging to the major context *Attack*, is the last part of the mission route, starting at a distance x from observed enemies in the objective area associated with the mission. The actual distance used in the experiments is defined in table 7.2. If there are no observed enemies in the objective area, major context *Attack* will not be a part of the plan, i.e. it will be pruned away in step 2 of the planning process. The remaining part of the route, between *Move* and *Attack* belongs to major context *Move Cautiously*.

5.4.6.2 Reactive behaviour

When unplanned events occur that forces an agent to deviate from a plan, other transition rules than "proceed with plan" will fire. When a transition rule fires, the agent's active context is replaced by the new context provided by the rule. This new major context is added first in the agent's plan. If the previous active context was *completed*, it is deleted from the plan, if not, it is kept as the next context. If the previous active context is kept, its status is changed to *not initialized*, so that the actions the agents should take when entering the context is repeated when the agent re-enters.

The way major contexts are added to an agents plan as a part of reacting to unplanned events could be interpreted as re-planning. However, the remaining part of the original plan is still a part of the new plan. Re-planning would mean going through the three planning steps again as the situation changes. The current implementation does not support this kind of re-planning, and there is no mechanism for checking whether the plan still is valid, i.e. will fulfil the mission context, as the situation changes.

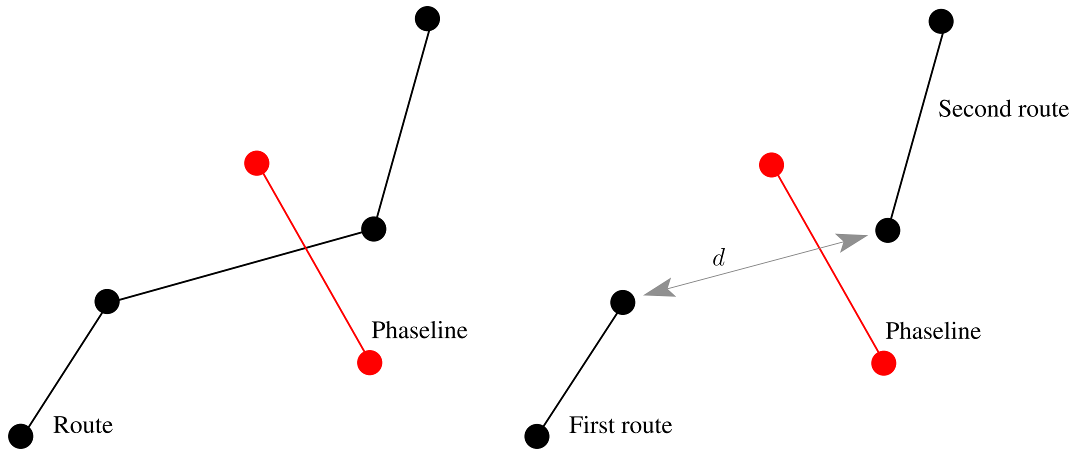
5.4.7 Splitting of routes

Both during planning and when continuing a route after an interruption we might need to split a route. This is done in the class `Route`, and the algorithms are described next.

5.4.7.1 Splitting of routes during planning

In the current version a set of detailed routes are sent from the C2IS system to the MAS. The set of routes contains one route for each platoon. As explained, each route may be divided into a set of subroutes during the planning process, each subroute connected to a context in the platoon's plan. The routes are split to fit the contexts in two cases, either by splitting the routes when they cross specific phase lines, as shown in Figure 5.3, or by splitting at the position where the route comes nearer than a given distance to a specific position, as shown in Figure 5.4. These two cases are implemented slightly differently.

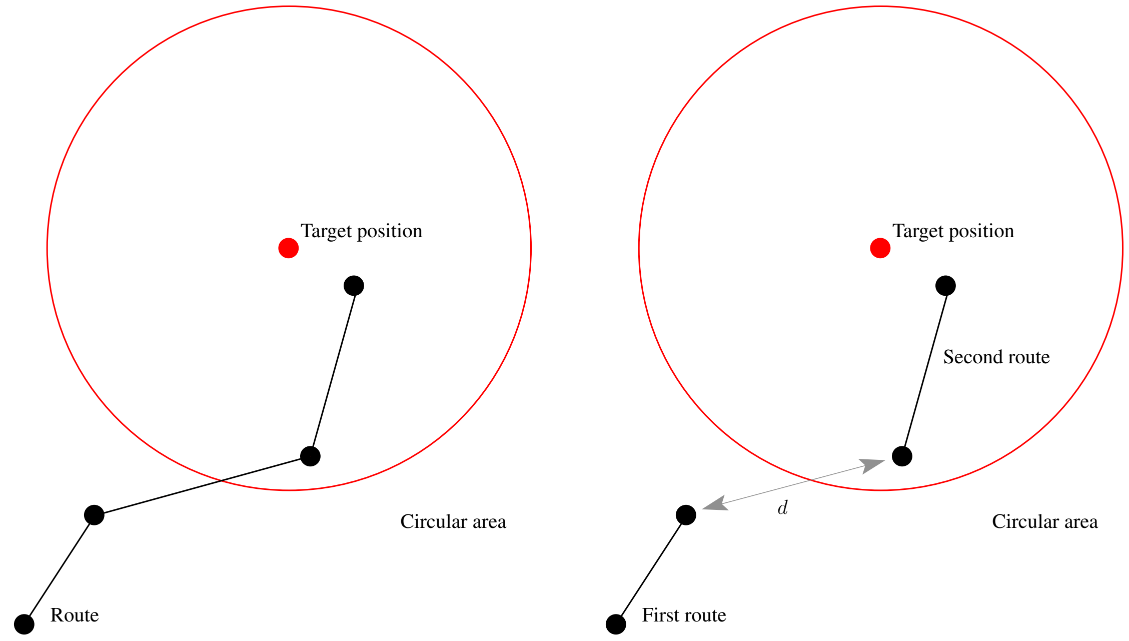
A route consists of an array with locations in the order they are to be visited, and a phase line consists of two locations. When a route is to be split when crossing a phase line, we propagate through the array, checking whether each line segment of the route crosses the phase line. After determining which line segment that intersects the phase line, we split the route into two subroutes. The first part contains all locations before the intersection point, while the second part contains all locations after the intersection point.



(a) A route crossing a phase line.

(b) The route is split into two new routes.

Figure 5.3 A route is split into two subroutes if it crosses a phase line.



(a) A route entering the circular area.

(b) The route is split into two new routes.

Figure 5.4 A route is split into two subroutes if it enters the circular area of a given radius around the target position.

When a route is to be split when it comes nearer than a given distance to a specific position, we construct a levelset function. This function is zero in all positions on the circle with the exact distance to the specific position, negative at positions further away and positive at positions closer to the specific position. We evaluate the levelset function at all locations in the route, and when the sign of the function value changes, the route crosses the circle and is to be split. All locations before the route intersects the circle are inserted in one route, while the remaining locations are inserted in a second route.

When the original route is split into two subroutes, we check the distance between the last location in the first route and the first location in the second route, shown as d in Figures 5.3b and 5.4b. This is to ensure smooth transitions when a platoon switches from the first to the second route, in particular if it at the same time changes formation. If the distance is too long, a change of formation is done over a long distance. To ensure a suitable distance between the two routes, we insert extra locations in either one of or both routes if the distance is too long. The limits for accepted distances are based on the distance from the first entity to the last one in the possible platoon formations, see section 6.2.2. The maximum length for a formation change should be longer than half the length of the longest formation, but not significantly longer. The longest formation is the custom column formation shown in Figure 6.5a, with a length of 150 m, and the maximum formation change distance is set to 100 meters.

5.4.7.2 Computing the remaining part of a route

If an entity during the simulation has deviated from its route, for example when attacking an unexpected enemy, we want it to afterwards continue following its original route. Retasking the entity to follow the original route in some cases causes the entity to move in the wrong direction to reach a node on the route before starting to move along the route. We therefore implemented a method that computed the remaining part of its route and tasked the entity to follow the remaining route. This method is illustrated in figure 5.5, where the red circle represents the entity, and the route is represented as black nodes connected by edges. We first compute the point along the route that is the nearest point to the entity, marked with a grey circle. Thereafter we determine which of the nodes that is the first after the grey circle. This is node $\mathbf{p}_3$ in the example. The last nodes along the route starting from this node form the remaining route, which in the example consists of $\mathbf{p}_3$ and $\mathbf{p}_4$.

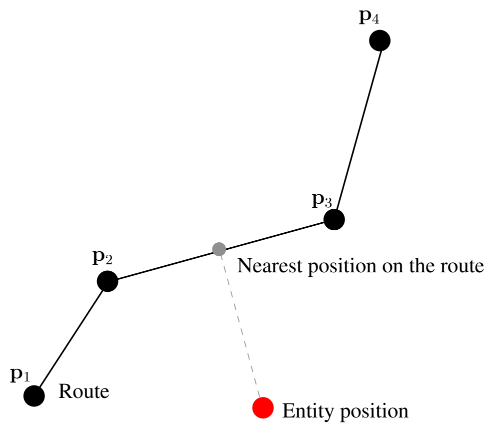


Figure 5.5 Computation of the remaining part of a route.

6 Configuration and extensions of VR-Forces

The CGF entities were simulated in VR-Forces. Section 6.1 describes the extensions we made to VR-Forces in order to control the entities from the MAS, and section 6.2 explains the customizations we made to the models in VR-Forces.

6.1 Low-level BML plug-in

Table 6.1 lists the Low-level BML message set that was implemented in the Low level BML plug-in. The messages consist of the message type and additional parameters. The messages were serialized to binary form with the use of Google Protocol Buffers (ProtoBuf) and transmitted between the MAS and the Low-level BML plug-in using HLA and the **ApplicationSpecificRadioSignal** interactions of the RPR-FOM.

Extensions and customization of the VR-Forces front-end and back-end can be made by either using the VR-Forces framework to create a custom application or by having VR-Forces load custom plug-ins at runtime. For our purposes it was sufficient to use the plug-in architecture of

Scenario Management	**Orders**	**Reporting**
Create area	Set concealed posture	Fired upon
Create aggregate	Move along route	Force assessment
Create aggregate from entities	Move into formation	Spot reports
Create entity	Reporting responsibility	Task status
Create phase line	Rules of engagement	
Create route	Wait	
Tick		

Table 6.1 Listing of the Low-level BML message set.

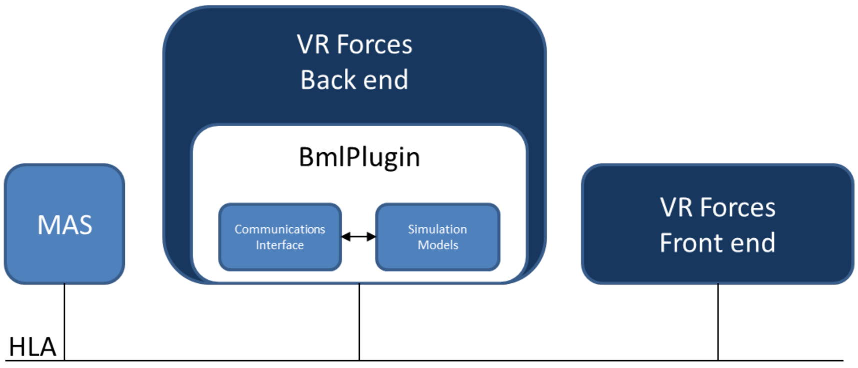


Figure 6.1 Federation view of the Low-level BML plug-in.

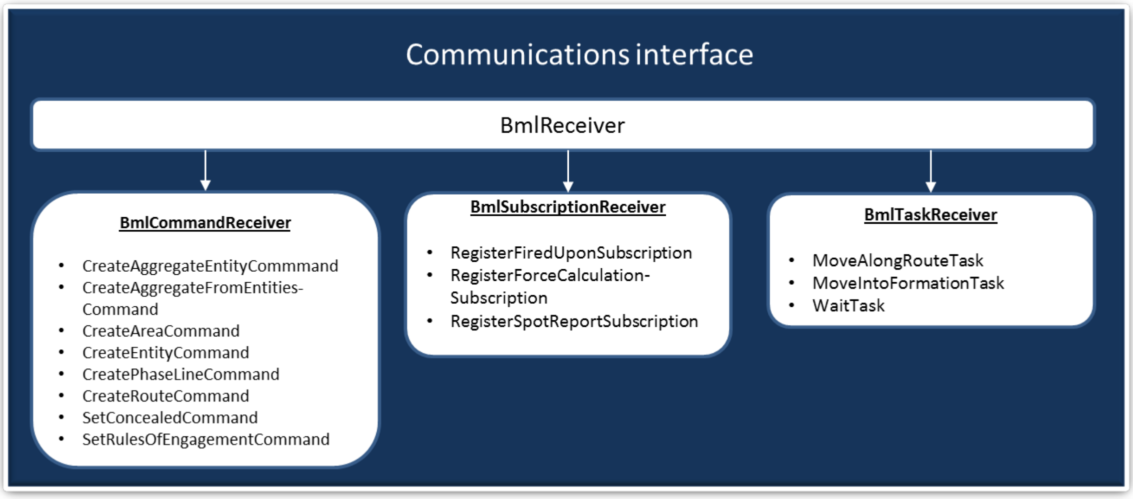


Figure 6.2 Communications interface of the Low-level BML plug-in.

the VR-Forces back-end to augment simulation models and extend the communications interface with Low-level BML. We hereby refer to our VR-Forces extensions as the Low-level BML plug-in. Figure 6.1 depicts the relationship between the HLA-based components.

6.1.1 Low-level BML plug-in design

The Low-level BML plug-in is loaded by the simulation engine when VR-Forces is started. All of the different components are however not instantiated straight away. The Communications interface is started together with the simulation engine. It can therefore accept commands immediately. The current commands, tasks and subscription requests have no meaning without a scenario and will therefore be ignored until one is manually loaded by the operator. In the future one could consider adding e.g. support for commands that can load scenario and terrain data. Figure 6.2 depicts the Communications interface component.

Messages from the MAS are received by the `LowLevelBmlReceiver` and routed to the

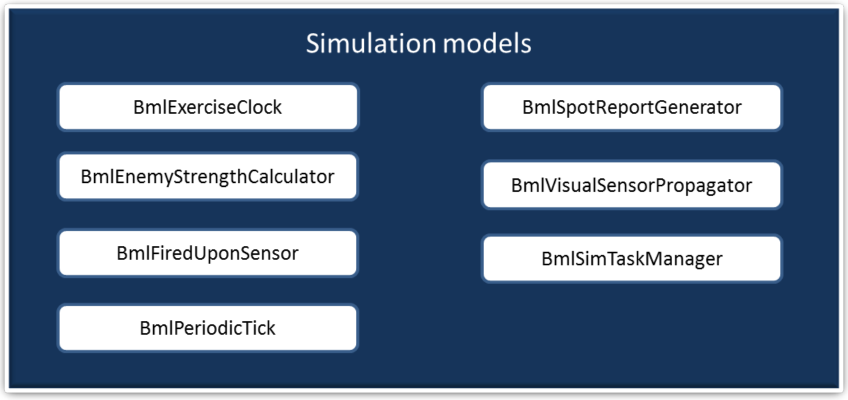


Figure 6.3 Simulation models implemented in the Low-level BML plug-in.

appropriate handler. The handlers are implemented as standalone classes with a well-defined interface, and it is straightforward to implement new handlers and register these with the `LowLevelBmlReceiver`.

Some of the commands e.g. `RegisterSpotReportSubscription` interact directly with the model implementation of e.g. Spot Report Generators that are attached to entities with sensors. The models are independent of the Communications interface, but offer an API that the handlers can use. The models publish messages to the MAS directly through HLA without using the Communications interface. The implemented simulation models are depicted in Figure 6.3. The simulation models are loaded together with the VR-Forces scenario and instantiated together with the entities they are part of. The only model that is always instantiated even in an empty scenario is the `BmlExerciseClock` which keeps track of simulated time.

6.1.2 Low-level BML message set definition

The messages that are sent between the MAS and the Low-level BML plug-in are defined in the ProtoBuf format. There are several reasons for using ProtoBuf. Firstly, defining and maintaining message definitions is easier and more readable in ProtoBuf format than with HLA FOMs. Secondly, ProtoBuf has a very efficient serialization engine with regards to both speed and the size of the serialized data. Thirdly, the ProtoBuf library can automatically generate the source code necessary to (de-)serialize the data for most popular programming languages, including Java and C++. The last point was decisive. While evolving the Low-level BML it was found valuable to be able to quickly generate (de-)serialization code for both the MAS and the CGF tool. The code generator for VR-Forces was not able to generate C++ code for our FOM extensions at that time. As a bonus backwards compatibility with DIS has been maintained.

A complete example of the message definition of the `CreateEntityCommand` is shown in listing 6.1. The message definition starts with the keyword `message` followed by the `name` of the message. Attributes are defined by the combination of a keyword that defines whether the field

```
message CreateEntityCommand {
  required string entityName = 1;
  required WorldLocation location = 2;
  required EntityType entityType = 3;
  required ForceType forceType = 4;
  optional double heading = 5;
}
message WorldLocation {
  required double x = 1;
  required double y = 2;
  required double z = 3;
}
message EntityType {
  required int32 kind = 1;
  required int32 domain = 2;
  required int32 country = 3;
  required int32 category = 4;
  required int32 subCategory = 5;
  required int32 specific = 6;
  required int32 extra = 7;
}
enum ForceType {
  Other = 0;
  Friendly = 1;
  Opposing = 2;
  Neutral = 3;
}
```

Listing 6.1 A complete example of the message definition of the `CreateEntityCommand`.

is `required`, `optional` or `repeated` (a list), a `type` and the `name` of the attribute. The `number` at the end is a unique identifier of the attribute within the message.

The messages are strongly typed. Primitive types such as integers, doubles and strings are directly supported by ProtoBuf. Enumerations and complex types can be defined as well as shown by the `ForceType` and `WorldLocation` definitions. The complete message definition file is then passed to a code generator that creates (de-)serialization code that can be compiled together with your own application. ProtoBuf supports many programming languages. We used ProtoBuf for both Java (MAS) and C++ (VR-Forces).

6.1.3 Simulation model improvements

The models and other improvements that were implemented in the Low-level BML plug-in are described below.

6.1.3.1 BmlExerciseClock

The exercise clock in VR-Forces manages the progression of time in the simulation. Elapsed simulation time is the amount of simulation time that has passed since the simulation started. The clock advances every time its tick() function is called.

To enable VR-forces to use scaled real-time with constant time steps (constant simulated time advance for every call to tick()) we developed a replacement for the standard exercise clock `DtExerciseClock`. We modified the `Fixed-Frame Best-Effort` mode explained in section 3.5.4 by dividing the sleep time with the factor given by the time slider in the VR-Forces front-end. Thus the pace of the simulation is increased if the computer is able to run the simulation faster.

It is preferable to have a constant time step to reduce the risk of having some of the simulation models become unstable. The standard `Variable-Frame Run-To-Complete` mode that VR-forces recommends for distributed interactive simulations has a tendency to break the simulation if the CPU becomes saturated. This happens because the time advance becomes too big. Until now it has been the only mode that supports scaled real-time without using HLA time management.

To load the `BmlExerciseClock` a custom `BmlSimManager` class was created. The `DtSimManager` class or derivatives thereof are responsible for instantiating all the classes needed to run a VR-forces application. They also configure the application correctly in order to be able to run a simulation.

6.1.3.2 Spot report generator

The standard `DtSpotReportGenerator` class is responsible for sending spot reports over HLA to other VR-forces back-ends as well as for visualization in the front-end. The format of the spot report is undocumented and is can only be directly accessed by other C++-based applications through the VR-forces Remote Control API. To be able to access the information in the spot reports from the MAS we created a subclass overriding the method sending the spot report to force VR-forces to send out the information over Low-level BML. The `BmlSpotReportGenerator` is loaded instead of the `DtSpotReportGenerator` for every entity with reporting capabilities. The `BmlSportReportGenerator` has an interface to allow the MAS to turn on and off spot reports for every entity through the Low-level BML plug-in Communications interface.

6.1.3.3 Periodic tick

When the *Low-level BML plug-in* is loaded, it instructs the VR-forces back-end to call a function in the `BmlPeriodicTick` class every second of simulated time. The `BmlPeriodicTick` class will then forward the current simulation time to the MAS. Time can only advance when a scenario is loaded.

6.1.3.4 SimTaskManager

The MAS depends on receiving status updates for the tasks it has issued to the entities. The `DtSimTaskManager` is responsible for managing the tasks of an entity. By creating a subclass, `BmlSimTaskManager`, and overriding the functions that are responsible for handling the tasks we are able to intercept the internal task notification messages and report back to the MAS whether the task has been completed, aborted or is still being executed. The `BmlSimTaskManager` class also has an API for registering task notification requests from the Low-level BML plug-in Communications interface.

6.1.3.5 FiredUponSensor

Every entity is equipped with a sensor which is implemented by the `DtFiredUponSensor` to notify the controller class that it is under fire. By creating a subclass, `BmlFiredUponSensor`, and overriding the function that notifies the controller that the entity is under fire we are able to send a fired upon message to the MAS.

6.2 Use and customization of models in VR-Forces

The customization of VR-Forces was kept to a minimum. We mainly used the models already included in the framework, even though they were simpler than what we would have liked. The few customizations and configurations that we did is explained next.

6.2.1 Entity models

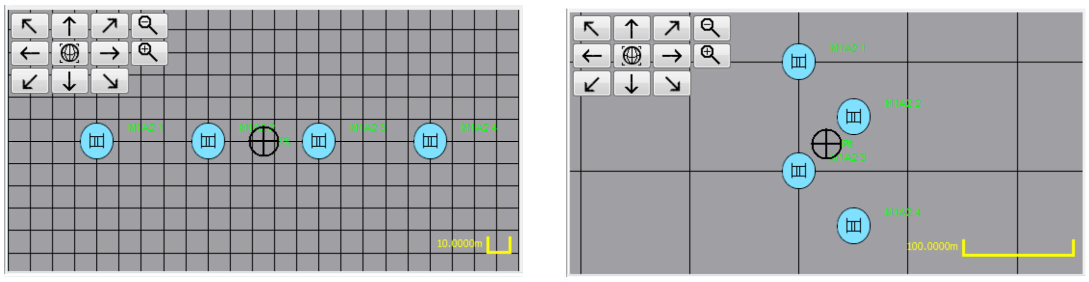
VR-Forces provides a standard simulation model set, which contains a set of standard entity models. In addition to the standard entities in VR-Forces, we have added models for the Norwegian unit types CV90 and Leo2, which have been used in previous projects. As we recall from section 3.5.1, entity models consist of sensors, controllers and actuators. Except from the 3D models of the entities, the Norwegian CV90 and Leo2 are constructed by suitable combinations of the standard entity components in VR-Forces. The entities are equipped with the standard visual sensor, and with weapon systems selected from the standard components.

The visual sensor models an entity's ability to observe its surroundings, and it has a set of properties, including its range for detection and its position on the entity. The entities we used in our experiment are listed in table 6.2, and all of these were equipped with only a visual sensor. The standard range of the visual sensor is 4000 m, and since forest is not accounted for, this distance may be considered too long in the area in our experiment.

A weapon system typically consists of controllers and actuators, and a set of connections describing how the controllers and actuators are linked. The weapon systems attached to an entity depends on the number and types of weapons the entity is equipped with. Each weapon system describes the properties of the weapon, like range, penetration abilities and the amount of damage it causes. The attach point of the weapon on the entity is also set, to calculate the trajectories when

	Entity	**Weapon System**
Blue Forces	CV90	30 mm gun
	Leo2	120 mm gun and M2 machine gun
Red Forces	T-72	125 mm gun
	BTR-80	M2 machine gun
	MT-LB	M2 machine gun

Table 6.2 The weapon systems used by the entities in our experiment.



(a) Standard line formation. (b) Standard column formation.

Figure 6.4 Two of the standard formations in VR-Forces.

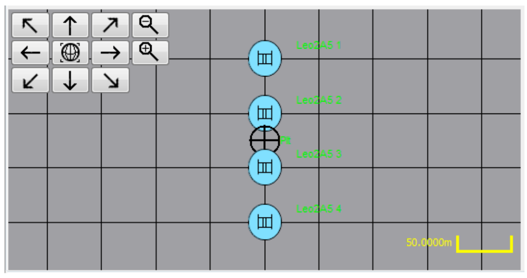
the weapon is fired. The types of weapons used by the entities in our experiment are shown in table 6.2. These weapons are standard in VR-Forces, and each weapon has parameters that can be varied. For example the limits for elevation range, the hit probability, and the load time can be adjusted.

It is possible in VR-Forces to set the parameters more accurately for each entity. This requires knowledge of better parameter values, which may be classified information.

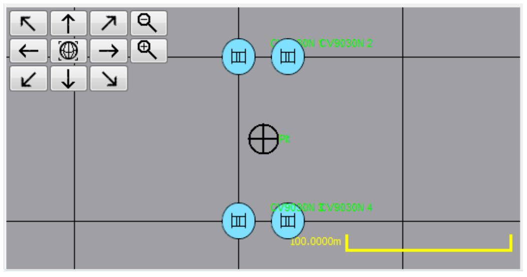
6.2.2 Formations

Two of the standard VR-Forces formations are shown in Figure 6.4. Our custom formations are shown in Figure 6.5. We use the standard line formation in figure 6.4a, but have customized the column formation as shown in Figure 6.5a, since this formation is more suitable for narrow Norwegian terrains and for movement on roads. In the line formation the distance between the entities is 50 m, and the width of the platoon is therefore 150 m. In the customized column formation the distances between the entities is set to 50 m, and the length of the platoon in this formation is 150 m. In addition we have added the formation two columns shown in Figure 6.5b. The width of this formation is 30 m and the length is 100 m.

A curiosity in VR-Forces is that if a platoon is tasked to move into formation in a specific position, the centre of the formation will be placed in this position, while when a platoon is tasked to follow a route using a given formation, it will first position itself such that the first entity (or the



(a) Custom column formation.



(b) Two columns formation.

Figure 6.5 Our custom formations.

midpoint of the front entities) is positioned at the starting point. The formation centres are marked as ⊕ in Figures 6.4 and 6.5. Therefore, when a platoon first is tasked to move into formation at the start point of a route and thereafter to follow the route, the platoon will first move into formation with the position at its centre, thereafter turn around and move back half the length of its formation, before it turns around to move along the route. In order to avoid this, we corrected the positions that were used for move into formation tasks by moving them back half the length of the formation. The length of the line formation is 0, so for this formation the positions were not moved.

The modelling of movement in formation is simple in VR-Forces. One entity is the leader of a formation, and the other entities will wait for the leader if it moves slowly, but if the leader moves too fast for the other entities to follow, the leader will not wait for the rest of the platoon. This is for example obvious if the leader is to the left in a line formation and the platoon takes a left turn. Then the leader has the shortest route to follow, and it leaves the remaining entities behind. This can partly be avoided by positioning the leading entity in the middle, but to ensure smooth movement, we need to implement a better algorithm.

6.2.3 B-HAVE

We chose not to use the B-HAVE plug-in. There were several reasons for this. First of all the path finding in B-HAVE does only compute the shortest path in the terrain, regardless of soil types and slope, which are important aspects of path finding in rural terrain. Therefore B-HAVE did not provide significant terrain analysis to the simulation, compared to the alternative, which was that VR-Forces let the entities follow the straight lines between specified locations. We also had some issues with B-HAVE, for example that there was a bug which made the entities gradually move to the right while moving forward. This bug occasionally caused the entities to move into the lake and get stuck. The bug was fixed in version 4.0.4i, which was released shortly before our demonstration. Because of the short time to the demonstration, and the fact that we had problems with VR-Forces crashing when using the new version of B-HAVE, we decided not to include B-HAVE in this version of the simulation system, but to consider it for later versions.

7 Experiment

The scenario we used to test our simulation system was an offensive military operation in an area near Alta. This scenario and a CxBR model suitable for the operation are described in detail in [42]. In this section we explain how this model is realized in the multi-agent framework, and the choices and adjustments we made in this process.

7.1 CxBR behaviour model

In [42] we described how to decompose an order through an hierarchy of agents, and we defined the major and minor contexts for battle command agents at company and platoon level. We also proposed a method for how the agents can plan their tasks, and defined how they should react to unplanned events.

We used the example scenario to limit the number of mission contexts and major contexts and transition rules. The content of these were all independent of the scenario, but only elements needed for simulating the example scenario had to be modelled and implemented. Detailed descriptions of the contexts will not be repeated here, but we will explain how we implemented the simplifications we had to make due to the lack of automated terrain analysis. This includes both planning of ordered task and reactive behaviour.

Table 7.1 lists all the missions and contexts we defined based on the example scenario. Both mission and major contexts were modelled independent of agent type (platoon or company), but not all of them are used for both platoons and companies. The table reflects which missions and contexts are used for which agents in the example scenario.

Figure 7.1 illustrates the full context map, with all possible transitions. The context map is valid for both platoon and company agents independent of mission context.

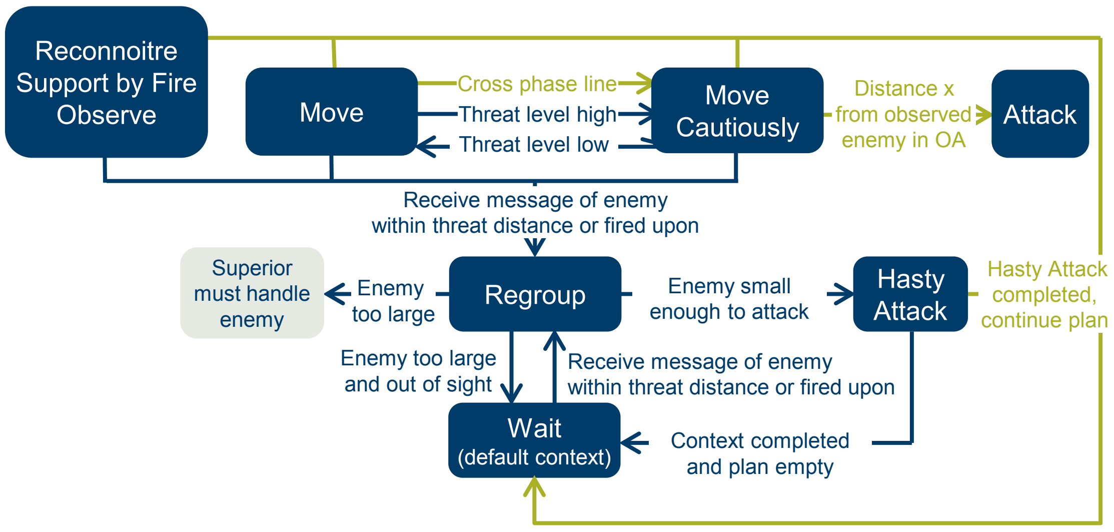
Some of the transition rules in figure 7.1 required further specifications during the implementation. For example, when is an enemy small enough to engage? How close to the enemy should the agent be before it transits to the *Attack* context? The parameter values used in the experiment are summarized in table 7.2.

7.2 Scenario in VR-Forces

Because of the lack of terrain analysis in path planning and identification of suitable strategic positions, we needed to provide accurate routes and positions to VR-Forces. These routes and positions were created in the C2IS and sent to VR-Forces, but in order to ensure that they were suitable for the scenario, we had first tested alternative routes and locations in VR-Forces before inserting the best routes and positions into the C2IS.

	Name	Used by Company Agents	Used by Platoon Agents	Used by CGF Agent
Missions Contexts	Reconnoitre	yes	no	no
	Seize	yes	no	no
	Support by Fire	yes	no	no
	Observe	no	yes	no
	Move	no	yes	yes
	Attack	no	yes	yes
	Wait	no	yes	yes
Major Contexts	Reconnoitre	yes	no	
	Support by Fire	yes	no	
	Move	yes	yes	
	Move Cautiously	yes	no	*CGF agents do*
	Attack	yes	yes	*not have contexts*
	Hasty Attack	yes	yes	
	Observe	yes	yes	
	Regroup	yes	yes	

Table 7.1 Mission and major contexts used to model the example scenario.



Reconnoitre
Support by Fire
Observe
Move
Cross phase line
Threat level high
Threat level low
Move Cautiously
Distance x from observed enemy in OA
Attack
Receive message of enemy within threat distance or fired upon
Superior must handle enemy
Enemy too large
Regroup
Enemy small enough to attack
Hasty Attack
Hasty Attack completed, continue plan
Enemy too large and out of sight
Receive message of enemy within threat distance or fired upon
Wait (default context)
Context completed and plan empty

Figure 7.1 The context map includes all company and platoon major contexts and is applicable for all company and platoon mission contexts.

	Parameter	**Value**
Planning	Distance from enemy to start *Attack* in mission *Seize*	700 m
Transition Rules	Enemy small enough to engage	Enemy force < Own force
	Enemy too large to engage	Enemy not small enough to engage
	Threat distance	700 m
Force Values	Force CV90	50
	Force Leo	100
	Force T-72	90
	Force BTR-80	30
	Force MT-LB	20
	Force Platoon/Company agent	Sum force subordinates

Table 7.2 Specifications of values used in the realization of the CxBR model.

Category	**Classifications**
Road	Dirt road, Asphalt
Water	Ocean, Shallow stream
Other areal features	Forest, Grass, Swamp, Building

Table 7.3 The terrain types present in the terrain used in the experiment.

7.2.1 The terrain in the scenario

The scenario takes place in an area near Alta. The synthetic terrain of this area used in the experiment has a resolution of 10 m and is created from a DEM 10 file provided by Kartverket. The vector data, which contain the specifications of the terrain types, are based on culture data with a resolution of 1:50 000. These vector data include roads, areal features and buildings. The VR-Forces terrain types that are used in the terrain model are shown in table 7.3.

The terrain type Shallow stream was chosen for the lakes and ponds, since it, as the only water alternative, has default values that makes it uncrossable. This was particularly important since we planned to use B-HAVE for path finding, and in order to avoid B-HAVE to plan routes crossing the water features, they needed to be uncrossable. We could also have modified the parameters for other water alternatives, but chose to use the one with suitable standard properties. The other terrain types were more obvious choices, there were not several suitable possibilities, for example not several types of forest.

7.2.2 Testing of routes and positions

We constructed a simple version of the scenario in VR-Forces, including the routes and the formations of the platoons, their rules of engagement and an attack of an enemy. The attack was in this test scenario performed according to the VR-Forces model of attack, which means to shoot at the enemy as soon as the enemy is observed and within weapon range. This scenario was used to create routes where the entities could not be attacked, and to plan how to move during the final attack of the enemy in order to exploit the cover possibilities of the terrain and attack the enemy from several angles simultaneously. The test scenario also helped us find routes that were of suitable lengths in order to avoid that one platoon had to wait for a long period of time while the other platoons reached specific positions. Trained military officers would have identified suitable routes faster than we did, but for untrained planners like us, small test simulations were particularly useful. The main plan for the operation was created with the help from Subject Matter Experts (SMEs), but the details of routes and locations were decided by us.

One of the reasons that we needed to test and plan the routes thoroughly, was that the models in VR-Forces do not consider reduced sight through forest. Since there is a significant amount of forest in the area of the operation, we needed to compensate for the lack of reduced sight through forest by instead placing the routes such that they were covered by terrain formations.

7.2.3 Red forces and their behaviour

With the help from SMEs, we chose the red forces shown in 7.2 for our experiment. These forces represent a vanguard for a main red force positioned further to the north east, and their task is to observe approaching blue forces and to slow them down such that the main red force can prepare their defence. The red forces are therefore rather weak compared to the blue forces, which are planning to attack the main red force, with support from an additional company, after attacking the vanguard.

The choice of entity types is based on the set of entity models included in VR-Forces. The red forces consist of three BTR-80, two T-72 and three MT-LB. The BTR-80 entities are positioned to overview the road and the surrounding area, searching for blue entities approaching from the south-east. The two T-72 are the main attack units of the vanguard and are positioned further back, ready to attack blue forces when they approach. The three MT-LB can be a small backup for the other entities, since they are hidden, the approaching blue forces may not observe them right away. They may also move along Route 1 around the lake in figure 7.2 in order to reconnoitre and search for approaching forces, or to move behind approaching blue forces. In our experiment we let the MT-LB move along the route searching for approaching enemies.

Since the red forces form a vanguard, they are not meant to stay and fight till the end, but to retreat after fulfilling their task of detecting and slowing down the approaching blue forces. They are therefore tasked to retreat along Route 2, which exits figure 7.2 in the upper right corner, moving towards the main red force.

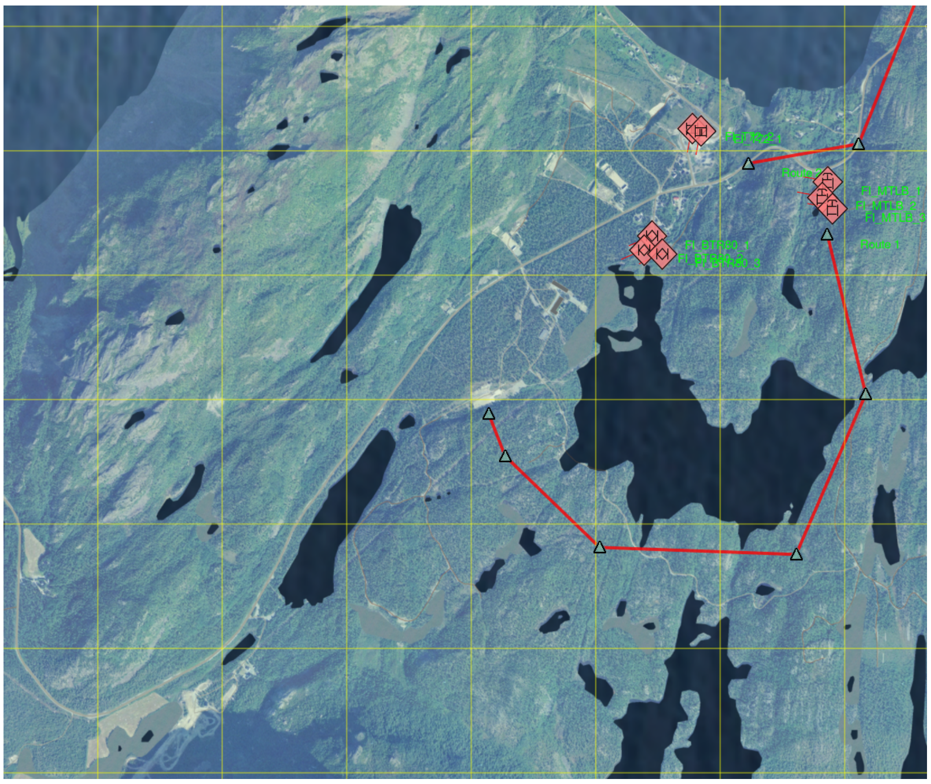


Figure 7.2 The red forces in our experiment.

8 Results

At November 14th 2012 we invited military experts from all branches of the military to a seminar where we demonstrated our simulation system. In this section we summarize the feedback from the participants.

8.1 Possible applications and desired functionality

It was a general consensus that a simulation system able to simulate an operational order would be useful, both for training and planning purposes. The discussions revealed possibilities that we had not thought about. For example, the system can be used to detect elements and factors that a plan depends on. This can for example be estimated use of fuel and ammunition, or if a helicopter is necessary for the plan to succeed, the plan will fail when fog or other weather conditions prevent the helicopter to take off. Elements that are necessary for a plan to succeed are important to detect, and, if possible, to avoid. If the necessary elements are missing, the plan needs to be changed.

For the simulation system to be useful there are however several factors that must be fulfilled. First, it must be straight forward to use and understand, and it should not be time consuming to perform a simulation. Second, the behaviour of the simulated entities must either be very realistic or the simulation must be only a simple visualization of the plan in time and space. Both these extremities might be useful, but everything in between would be useless.

During the demonstration we explained the simplifications we had had to make due to the lack of automatic terrain analysis. For example we had to draw routes in the C2IS. Based on the feedback, it should be possible to use the system to compare alternative routes, for example on different sides of a lake, during the planning of an operation. It should therefore be possible for an officer to provide guidelines for a route in the C2IS, such that these guidelines are followed during path planning in the simulation system. If no guidelines are provided, the path planning should be based on terrain information alone.

As for the behaviour of the red forces, they should always follow doctrine. In war the enemy forces may come up with surprising tactics and unexpected solutions, but when simulating the plan one will want the red forces to behave according to standard doctrine. The simulations should be used to prepare for standard enemy behaviours. A simulated enemy can never cover all possible human solutions, and it should stick to doctrine instead of attempting to be smart.

8.2 Comments on the current behaviour

In addition to comments about desired functionality, we got some comments on the behaviour of the simulated entities.

In the simulation of the offensive operation, a company task is not started before the previous is completed. This is due to the synchronization matrix we used, which stated that the tasks should begin when the previous tasks were finished. It was commented that normally a company starts its task before the previous company is completely finished. This can be done for example by introducing report lines. When a company crosses the report line, it sends a radio message which triggers the next company to start its task. It is also possible to let each task be scheduled to start at a specific time.

Our agents were tasked to continue attacking the enemy until all entities were destroyed. Military personnel commented that it is normally not necessary to destroy all enemies in order to defeat the enemy. The structure of the enemy forces needs to be sufficiently reduced, and the speed of the forward movement as opposed to the need to destroy small enemy entities needs to be considered. In our simulation system we need to determine limits for the number of entities that can be left when the friendly forces continue their plan, depending on the strength of the friendly forces and their need to proceed forward quickly.

9 Conclusions and future work

In this report we have described a demonstrator of a simulation system which can autonomously simulate an order from a C2IS. The focus so far has been on making the systems work together and implement a multi-agent framework based on CxBR. The MAS includes enough behaviour to demonstrate the concept and to reveal new challenges and research questions. In section 9.1 we have gathered considerations that need to be discussed further, and in section 9.2 we discuss changes and extenuation we intend to make.

9.1 Topics for discussion

Many decisions regarding how the simulation system should work, capabilities, limitations, stochastic versus deterministic, etc. depends on what the simulation system should be used for. We should compare with GESI and VBS2 that are used of specific types of simulations in order to determine which properties we want for our simulation system, and which functions we do not need or want. We need a sufficiently good simulation, but we need to determine the limit for sufficiently good and the requirements for the simulation.

A simulation can simulate aggregated units, vehicles and/or single soldiers. At the moment VR-Forces receives tasks at platoon level and tasks single vehicles according to the behaviour models in VR-Forces. The vehicles are visualized in VR-Forces, while the aggregated units are visualized in the C2IS. Whether we want to keep this level of simulation, simulate single soldiers, or only simulate aggregated units can be discussed further. The choice will depend on the use of the simulation system. For planning purposes a simulation of aggregated units is probably sufficient, while for simulated exercises where soldiers are trained by controlling entities in a synthetic environment, vehicles, and maybe also single soldiers, should probably be represented.

The visualization of the simulation should be discussed further, not only which level of entities that should be represented, but also the appearance of the visualization itself, regarding maps, symbols, effects, and information. The visualization should make the simulation easy to understand, be visually appealing, and include useful and necessary information.

The requirements for the quality of the simulated behaviour need to be determined. Whether the simulation system only shall follow detailed provided orders or include more intelligent behaviour, place different requirements on the artificial intelligence required in the simulation system.

There are several possibilities for what types of maps that should be used in the simulations. The maps can be streamed, for example from military sources, and the simulation may be performed on top of the chosen map. This would give a simple, but interesting visualization, with the flexibility that the type of map can be chosen. The maps can also be stored locally.

The simulation system can be extended to include logistics, for example use of ammunition and fuel, and how and where new resources should be received. There are also other possible extensions of the simulation system, and it can be discussed further which extensions we want and which that are out of the scope of our simulation system.

9.2 Future work

There is some desired functionality that is not yet provided. Most importantly, automatic terrain analysis is missing in the current version. We are working on improving the terrain analysis in B-HAVE. A preliminary study can be found in [43]. Weather conditions and their implications are also interesting to include in future versions.

As for the multi-agent system we need more cooperation and communication between the entities.

For example when one platoon performs a hasty attack on a small enemy it encounters, it does not report this to its superior. It is therefore a possibility that several platoons initiate a hasty attack on the same enemy. Instead the platoon should report the initiation of a hasty attack to its superior, such that when the second platoon reports initiation of a hasty attack, the company tasks the platoon to instead proceed with plan. It is also reasonable that the superior agents always keep track on what their subordinate agents are doing.

Another desired extension of the multi-agent framework is the ability to re-plan. In the current version of the MAS, the agents make a plan for the task when they receive it based on the situation in that moment. The agent will react to changes in the environment, like an unexpected enemy, but will not notice if the initial plan is no longer valid, or whether it would have planned differently with the updated picture of the situation.

In addition to the ability to do re-planning, the planning procedure itself must be improved. In section 7.1 we explained how a plan is made for the mission context *Seize*. The basic plan lacked considerations like potential enemies that had been observed outside objective areas.

To make the simulation system useful, we do not only need more complex basic plans for the mission contexts which are currently available, but we will need support for more tasks. Also, the simulation system needs a GUI where the user can monitor the commands, reports, mission contexts, major contexts etc., and possibly interfere with the simulation when there is a need to adjust the entities' behaviour. Such a user interface will make it easier to understand the decision making done by the agents, and should produce a log for later analysis.

References

[1] S. Carey *et al.*, “Standardizing battle management language - a vital move towards the army transformation,” in *Proceedings of the 2001 Fall Simulation Interoperability Workshop*, no. 01F-SIW-067, 2001.

[2] R. Bronkers *et al.*, “Battle management language capable computer generated forces,” in *Proceedings of European Simulation Interoperability Workshop*, 2011.

[3] A. Alstad *et al.*, “Low-level battle management language,” in *Proceedings of the 2013 Spring Simulation Interoperability Workshop*, no. 13S-SIW-032, 2013.

[4] A. J. Gonzalez, B. S. Stensrud, and G. Barret, “Formalizing context-based reasoning: A modeling paradigm for representing tactical human behavior,” *International Journal of Intelligent Systems*, vol. 23, pp. 822–847, 2008.

[5] A. Gallagher, A. J. Gonzalez, and R. DeMara, “Modeling platform behaviors under degraded states using context-based reasoning,” in *Proceedings of the 2000 Interservice/Industry Training, Simulation and Education Conference (I/ITSEC-2000)*, 2000.

[6] A. J. Gonzalez and R. Ahlers, “Context-based representation of intelligent behavior in training simulations,” *Transactions of the Society for Computer Simulation International*, vol. 15, no. 4, pp. 153–166, 1998.

[7] SISO, *Standard for: Coalition Battle Management Language (C-BML)*, 2012, SISO-STD-011-20128-DRAFT.

[8] S. Levine *et al.*, “Joint battle management language (JBML) - phase 1 development and demonstration results,” in *Proceedings of the 2007 Fall Simulation Interoperability Workshop*, no. 07F-SIW-051, 2013.

[9] A. Alstad, “Norwegian Nato MSG-048 technical documentation,” Forsvarets forskningsinstitutt, FFI-notat 2010/01261, 2010.

[10] SISO. Military scenario definition language (MSDL). [Online]. Available: http://www.sisostds.org/StandardsActivities/DevelopmentGroups/MSDLMilitaryScenarioDefinitionLanguage.aspx

[11] J. M. Pullen *et al.*, “MSDL and C-BML working together for NATO MSG-085,” in *IEEE 2012 Spring Simulation Interoperability Workshop*, 2012.

[12] NATO NSA, *STANAG 4603 - Modelling and Simulating Architecture Standards for Technical Interoperability: High Level Architecture (HLA)*, 2008.

[13] SISO, *Realtime-Platoform Reference Federation Object Model (RPR FOM 2.0d17)*, 2003.

[14] M. N. Nielsen and O. M. Staal, “Evaluering av high level architecture kjøretidsinfrastruktur for bruk ved FFI,” Forsvarets forskningsinstitutt, FFI/NOTAT-2006/03774, 2006, (Unntatt Offentlighet).

[15] IEEE, *Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Object Model Template (OMT) Specification*, August 2010, IEEE Std 1516. 2-2010 (Revision of IEEE Std 1516. 2-2000).

[16] IEEE, *Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Federate Interface Specification*, August 2010, IEEE Std 1516. 2-2010 (Revision of IEEE Std 1516. 2-2000).

[17] IEEE, *Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules*, August 2010, IEEE Std 1516. 2-2010 (Revision of IEEE Std 1516. 2-2000).

[18] IEEE, *Standard for Distributed Interactive Simulation - Application Protocols*, 2012, IEEE Std 1278.1-2012 (Revision of IEEE Std 1278.1-1995).

[19] M. Wooldridge, *An Introduction to MultiAgent Systems*. John Wiley & Sons Inc., 2002.

[20] VT MÄK VR-Forces. [Online]. Available: www.mak.com/products/simulate/computer-generated-forces.html

[21] Å. Hjulstad, “Simuleringsrammeverk for datagenererte styrker - erfaringer med og tilpasninger i et COTS-produkt: VR-Forces,” Forsvarets forskningsinstitutt, FFI/NOTAT-2005/01788, 2005, (Unntatt Offentlighet).

[22] VT MÄK Technologies, *VR-Forces Developers Guide*.

[23] VT MÄK Technologies, *VR-Forces Configuration Guide*.

[24] R. Lerusalimschy, *Programming in Lua*. lua.org, 2013.

[25] VT MÄK Technologies, *B-HAVE Users Guide*.

[26] Autodesk Kynapse. [Online]. Available: www.gameware.autodesk.com/kynapse

[27] Multilateral Interoperability Programme (MIP), *STANAG 5525 Joint Consultation Command and Control Information Exchange Data Model (JC3IEDM)*, 2012.

[28] M. J. Pullen *et al.*, “Adding reports to coalition battle management language for NATO MSG-048,” in *Joint SISO/SCS European Multi-conference*, 2009.

[29] M. J. Pullen *et al.*, “Integrating national c2 and simulation systems for bml experimentation,” in *Proceedings of European Simulation Interoperability Workshop 2010*, 2010.

[30] M. Pullen *et al.*, “Integrating national C2 and simulation systems for BML experimentation,” in *Proceedings of the 2010 European Simulation Interoperability Workshop*, no. 10E-SIW-008, 2010.

[31] M. Pullen *et al.*, “An expanded C2-simulation experimental environment based on bml,” in *Proceedings of the 2010 Spring Simulation Interoperability Workshop*, no. 10S-SIW-049, 2010.

[32] Google Protocol Buffers. [Online]. Available: https://developers.google.com/protocol-buffers/

[33] R. M. Fujimoto, “Time management in the high level architecture,” *Simulation*, vol. 71, pp. 388–400, 1998.

[34] A. Alstad, “HlaLib v3.0 - user guide,” Forsvarets forskningsinstitutt, FFI/NOTAT-2010/0871, 2010.

[35] J. Gemrot *et al.*, “Pogamut 3 can assist developers in building ai (not only) for their videogame agents,” in *Agents for Games and Simulations*, ser. Lecture Notes in Computer Science, F. Dignum *et al.*, Eds. Springer Berlin Heidelberg, 2009, vol. 5920, pp. 1–15. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-11198-3_1

[36] A. Pokahr, L. Braubach, and W. Lamersdorf, “Jadex: a BDI reasoning engine,” in *Multi-Agent Programming - Languages, Platforms and Applications*, ser. Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer, 2005, vol. 15, ch. 6, pp. 149–174.

[37] G. C. Barrett and A. J. Gonzalez, “Effective agent collaboration through improved communication by means of contextual reasoning,” *International Journal of Intelligent Systems*, vol. 26, no. 2, pp. 129–154, 2010.

[38] Drools - the business logic integration platform. JBoss. [Online]. Available: www.jboss.org/drools/

[39] Websphere ilog jrules brms. IBM. [Online]. Available: www-01.ibm.com/software/integration/business-rule-management/jrules-family/

[40] Jess, the rule engine for the java platform. [Online]. Available: www.jessrules.com

[41] Clips: A tool for building expert systems. [Online]. Available: clipsrules.sourceforge.net

[42] R. A. Løvlid *et al.*, “Modelling battle command with context-based reasoning,” Forsvarets forskningsinstitutt, FFI-rapport 2013/00861, 2013.

[43] S. Bruvoll, “Preliminary study of terrain analysis and path planning for computer generated forces,” Forsvarets forskningsinstitutt, FFI-notat 2013/00688, 2013.

Abbreviations

ANNCP	Anglo-Netherlands-Norwegian Cooperation Program
C2IS	Command and Control Information System
C-BML	Coalition Battle Management Language
CBMS	Coalition Battle Management Service
CGF	Computer Generated Forces
COTS	Commercial Of-The-Shelf
CxBR	Context-Based Reasoning
DIS	Distributed Interactive Simulation
GMU	George Mason University
GUI	Graphical User Interface
HLA	High Level Architecture
IEEE	Institute of Electrical and Electronics Engineers
I/ITSEC	Interservice/Industry Training, Simulation and Education Conference
JMS	Java Message Service
KDS	Kongsberg Defence Systems
LOCON	LOwer COntrol operators
LVC	Live-Virtual-Constructive
MAS	Multi-Agent System
MOM	Message Oriented Middleware
MSG	Modelling and Simulation Group (in NATO)
MSDL	Military Scenario Definition Language
OA	Objective Area
OMT	Object Model Template
OPORD	Operational Order
ORBAT	Order of Battle
ROE	Rules of Engagement
SBML	Scripted Battle Management Language
SISO	Simulation Interoperability Standards Organization
SME	Subject Matter Expert
VMASC	Virginia Modeling, Analysis and Simulation Center
VR-Forces	Virtual Reality Forces
XML	eXtensible Markup Language (XML)