Battle Management Language

- An Implementation for a Military Scenario Editor

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg
Computer Science

Bachelor thesis:
Henric Lind
Mathias Lubera

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2009

Abstract

Battle Management Language - An Implementation for a Military Scenario Editor

In order to plan and control a military simulation, there is a need for a language which can be understood by all parts and levels of the organization - both man and machine. This need has encouraged the development of a highly structured and unambiguous language. The term for this type of language is *Battle Management Language*. The United States military has further developed the concept into an actual solution for use with joint forces, called *Joint Battle Management Language*.

Saab Training Systems AB is a company which develops training solutions for military training. Many of their solutions involve linking different systems with each other, and often with a simulation platform. *Saab Training Systems AB* has also developed *General Scenario Editor* - an application in which it is possible to plan a military operation by positioning military units on a battlefield. *Battle Management Language* will make it possible to control different units and state commander's intent with a universal language.

The goal is to enable the user to manage orders in *General Scenario Editor* by assigning tasks to involved units in the scenario. The application should be able to export a scenario to *Joint Battle Management Language* using *XML*.

General Scenario Editor is not intended solely for use with *Joint Battle Management Language*. The biggest challenge is to allow users to plan a military scenario without the concern of the destined output format. This means that *Joint Battle Management Language* cannot fully determine the design of the application. However, the concept of *Battle Management Language* can still influence the method for structuring and storing orders in the application. This is done with the *5 W's principle*, which can be recognized as the fundamental of all military tasks. The principle is used in the implementation and specifies *What*, *Who*, *Where*, *When* and *Why* for each military task.

The result is that *General Scenario Editor* can be used to plan a scenario, complete with assigned orders. The scenario can be exported to *Joint Battle Management Language*. In order to further show the proof of concept, a scenario can also be exported to the game *Virtual Battlespace 2*, in which the scenario can be visualized and simulated.

Keywords: Battle Management Language, BML, Joint Battle Management Language, Coalition Battle Management Language, General Scenario Editor

Sammanfattning

Battle Management Language – En implementation för General Scenario Editor

För att kunna planera och kommendera en militär simulation finns det behov för ett språk som alla delar av organisationen kan förstå – såväl människa som maskin. Behovet har drivit på utvecklingen av ett välstrukturerat och entydigt språk, vilket benämns som *Battle Management Language.* Militärväsendet i USA har nyttjat konceptet för att skapa en fungerande lösning att användas med sina förenade styrkor, kallat *Joint Battle Management Language*.

Företaget *Saab Training Systems AB* utvecklar träningslösningar för militära ändamål. Många av företagets lösningar innefattar ihopkoppling av olika system, ofta i samband med en simulationsplattform. *Saab Training Systems AB* har också utvecklat *General Scenario Editor* - en applikation som gör det möjligt att planera en militär operation genom att placera ut enheter på en karta. *Battle Managment Language* gör det möjligt att ge uppgifter till enheter med hjälp av ett allomfattande språk.

Målet är att kunna ge användaren möjlighet att hantera militära ordrar i *General Scenario Editor* genom att ge uppgifter till enheter i ett scenario. Applikationen ska kunna exportera ett scenario till *Joint Battle Management Language* med hjälp av *XML*.

General Scenario Editor är inte enbart till för att användas med *Joint Battle Management Language*. Den största utmaningen är att låta användaren planera ett militärt scenario i applikationen utan tanke i vilket system informationen skall användas. Detta innebär att *Joint Battle Management Language* inte ensamt kan bestämma hur applikationen ska vara uppbyggd. Däremot kan konceptet av *Battle Management Language* användas för att strukturera och lagra uppgifter i applikationen. Detta sker genom att man i detalj anger *Vad*, *Vem*, *Var*, *När* och *Varför* för varje militär uppgift. Detta kan ses som den mest väsentliga informationen för alla militära uppgifter.

Resultatet är att *General Scenario Editor* kan användas för att planera ett militärt scenario, med tillhörande militära ordrar. Scenariot kan exporteras till *Joint Battle Management Language*. För att ytterligare styrka konceptet, kan ett scenario exporteras till datorspelet *Virtual Battlespace 2*, i vilket scenariot visualiseras och spelas upp.

Nyckelord: Battle Management Language, BML, Joint Battle Management Language, Coalition Battle Management Language, General Scenario Editor

Foreword

This bachelor thesis was made at *Saab Training Systems AB* in Helsingborg for Lund University, LTH School of Engineering.
The main contribution of this thesis is the software extensions made to *General Scenario Editor* and the research around *Battle Management Language*.

Thanks to our examiner Lise Jensen at LTH and our expert advisors Per Gustavsson, Jakob Blomberg, Deepak Sumra and Stefan Lundmark for great guidance and aid. Also special thanks to other employees at *Saab Training Systems AB* for help and support.

List of contents

1 Background

In order to plan and control a military simulation, there is a need for a language which can be understood by all parts and levels of the organization - both man and machine. This need has encouraged the development of a highly structured and unambiguous language. The term for this type of language is *Battle Management Language*.

Saab Training Systems AB is a company which develops training solutions, mostly for military training. Such solutions often involve linking existing systems with a simulation platform. This integration between systems is significantly simplified by *Saab Training Systems AB* current solution, *WISE Connectivity* (see Appendix 9.1).

The development of a standardized *Battle Management Language* has now reached a point where *Saab Training Systems AB* is interested in the concept of such a language. Using *Battle Management Language* with *WISE Connectivity* will make it possible to control different linked systems and simulations with a common language.

1.1 Previous work by Saab Training Systems

To build military training scenarios that can be understood by different systems, the *Military Scenario Definition Language* standard was set by the *Simulation Interoperability Standards Organization*. While this language is merely used to define the organization and its resources of a military operation, *Battle Management Language* is intended for managing orders in the same organization. [4, 2]

In 2008 a student group at *Saab Training Systems AB* investigated the possibilities of *Military Scenario Definition Language*. The study, made by Fredrik Ullner and Anders Lundgren, led to a product implementing support for the language. This product is called *General Scenario Editor* and is currently under development. [5]

As an initial step to investigate the concept of *Battle Management Language*, *Saab Training Systems AB* now plans to further extend the scenario editor to include support for the new language.

1.2 General Scenario Editor

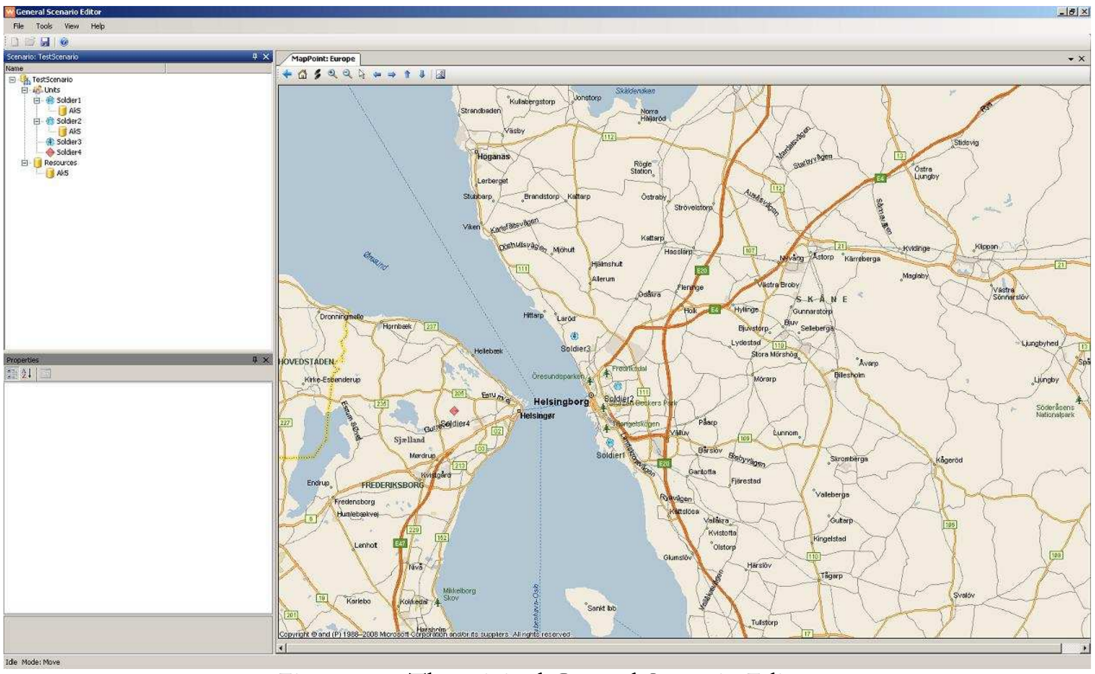


Figure 1-1: The original General Scenario Editor

As seen in Figure 1-1, the original version of *General Scenario Editor* presents the information using a map and a tree-view containing the units of the organization. The user can define the organization and place units on the map. The original version of the scenario editor have the possibility to import and export *Military Scenario Definition Language 1.0* using *eXtensible Markup Language*, also known as *XML*. There is only very minor functionality regarding order management.

1.3 Battle Management Language, BML

1.3.1 History

Command & Control is a term to describe the routine of a commanding officer to assign tasks to forces in a military mission. Ever since computerized *Command & Control* systems were introduced, there has been an increasing wish to use simulation systems in military training. Several systems have been used and have become more and more effective. This improvement has especially affected the training for the higher echelons in the military. [11]

The problem for computer-based training systems has been the large number of required personnel for controlling the simulations, which made this type of training very resource demanding. The main reason originated in the lack of means to communicate between different systems and the simulation. Improved communication between these systems would make it possible for different systems to exchange information and also to interact with the simulation. This would reduce the number of workstation controllers needed.

To achieve this, later systems evolved to rely on "free text"-messages within the *Command & Control* messages. Although communication with "free text"-messages is easily understood and structured by humans, computerized systems can merely distribute these messages, but not understand them.
To further develop intelligent systems that can analyze, interpret and control a battle situation, an unambiguous and structured language for orders has to be defined. This is where *Battle Management Language* comes into the picture. [11]

1.3.2 Concept

In the paper "Standardizing Battle Management Language – Facilitating Coalition Interoperability" by Hieb, we learn that a language for managing battle situations is actually nothing new. There has always been a language to direct forces in military operations. Nowadays, a commander must be able to structure an order so that it can be understood correctly by all participants, both human, machine and simulation. Hieb also means that early solutions, like *EAGLE BML* and *CCSIL*, did not successfully accomplish a generic solution to all problems. The major problem was that the mentioned languages were very dependent on the specific simulation in question and required a great amount of customization. [2, 11, 25]

As a result of the development of a standardized *Battle Management Language*, the following definition for BML has been set:

> ***"BML is 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."*** [20]

When *Battle Management Language* (BML) is used in this report, the above definition is the one assumed.

During the recent years there have been several attempts for creating a *Battle Management Language*. Some of them are more specific than others, and targeted for different purposes. The solutions listed below have been investigated to find an appropriate candidate to use explicitly in this thesis.

	Specification	Ground	Air	Naval	Implementation	Software services	International
MIP/JC3IEDM	X	X	X	X			X
ArmyBML		X			X		
JBML	X	X	X	X	X	X	
C-BML	X	X	X	X		X	X
geoBML	X	X			X		

Table 1-1: The table shows the specific purposes and abilities for some well known languages.

1.3.3 C2IEDM and JC3IEDM

To allow commanders to make fast decisions in joint operations, there must be an interoperability of information. Also, in operations which include several countries, the information needs to be exchanged across both national standards and lingual boundaries.

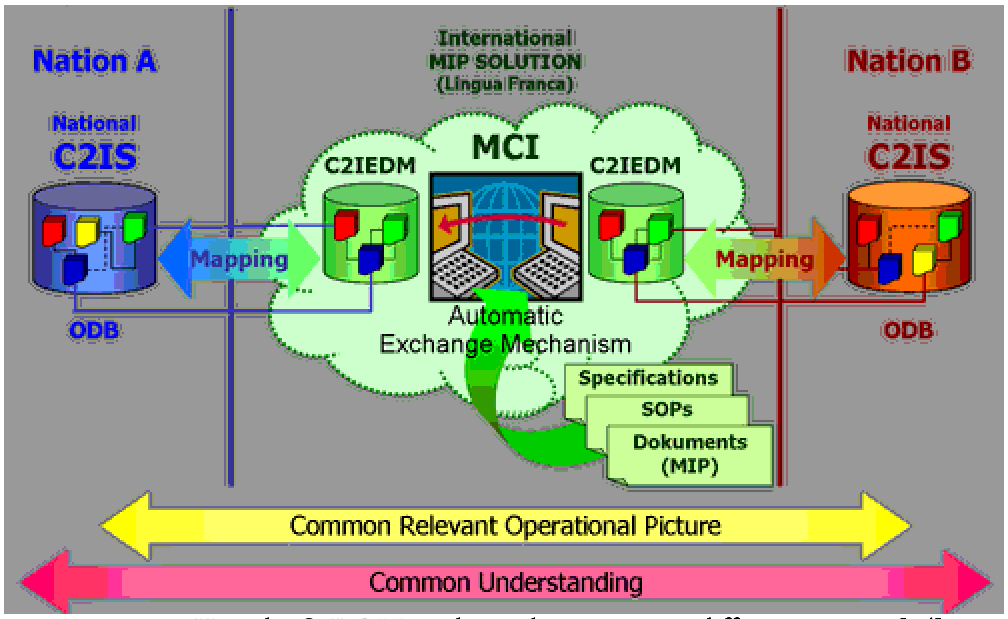


Figure 1-2: How the C2IEDM can be used to connect to different systems. [24]

Multilateral Interoperability Programme tries to achieve interoperability of *Command & Control* information systems for all levels of the organization. *Multilateral Interoperability Programme*'s solution is the *Command and Control Information Exchange Data Model*, or C2IEDM. The model structures the exchangeable information needed by the commanders in a joint operation. This allows these different systems to exchange information, and makes it possible to decide what and to whom the information should be exchanged. [7, 10]

The C2IEDM is an earlier version of the current *Joint Consultation, Command and Control Information Exchange Data Model*, or simply JC3IEDM. [3] C2IEDM and JC3IEDM are used as a base for some available solutions of *Battle Management Language*.

1.3.4 Army BML

As described in the paper "Evaluating the Proposed Coalition Battle Management Language Standard as a Basis for Enhanced C2 to M&S Interoperability", the military of the United States has made some attempts to develop a feasible *Battle Management Language* for use with their joint forces. The development started by studying many doctrine manuals; from very general field manuals to more detailed manuals for the different branches of the United States military. This study resulted in the definition of the 5 W's principle, which stands for *What*, *When*, *Where*, *Who* and *Why*. This principle can be used to structure a military task. More details about the 5 W's can be found later in chapter 5.1. In 2003 a prototype, *Army BML*, demonstrated the principles of *Battle Management Language* for battalion operations. This prototype led to a follow-up project, *XBML*, which was sponsored by *US Joint Forces Command*. The main goals for *XBML* was: [18]

1. To use web technology for communication between the systems.
2. Use the *Command and Control Information Exchange Data Model*.

1.3.5 JBML

US Joint Forces Command saw the potential of using *XBML* with *Command & Control*-systems and simulations and realized that its concept was feasible for both air and ground operations. This later led to the development of *Joint Battle Management Language* (JBML). [3]

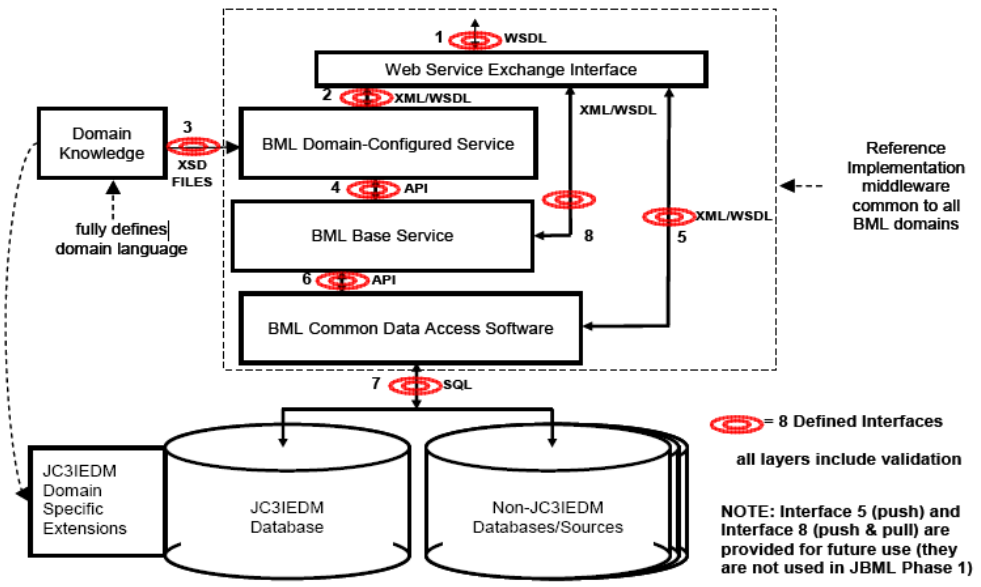

1 WSDL
Web Service Exchange Interface
2 XML/WSDL
XML/WSDL
Domain Knowledge
3 XSD FILES
BML Domain-Configured Service
4 API
fully defines domain language
BML Base Service
8
XML/WSDL
5
Reference Implementation middleware common to all BML domains
6 API
BML Common Data Access Software
7 SQL
= 8 Defined Interfaces
all layers include validation
JC3IEDM Domain Specific Extensions
JC3IEDM Database
Non-JC3IEDM Databases/Sources
NOTE: Interface 5 (push) and Interface 8 (push & pull) are provided for future use (they are not used in JBML Phase 1)

Figure 1-3: The different layers of JBML

JBML is characterized by three layers in its implementation. The above layer, *Domain-Configured Service*, defines *Battle Management Language* in a domain. It is implemented by a web service which uses *XML* to transfer all information needed by the 5 W's. This is from where the user of the application access JBML. [3]

The middle layer, *BML Base Service*, disassembles the 5 W's into information that can be placed in the corresponding tables in JC3IEDM. It uses an interface to the bottom layer, *Common Data Access Software*, to access these tables. [3]

1.3.6 C-BML

The *Coalition Battle Management Language* (C-BML) is a currently developing standard by *Simulation Interoperability Standards Organization*. Unlike *Joint*

Battle Management Language, *C-BML* only includes a specification, not an implementation. It is supposed to be a definition of how military tasks should be structured. [3, 8]

C-BML is built upon the 5 W's principle to translate military tasks to a subset of the *Command and Control Information Exchange Data Model*. In a paper by Tolk, it is revealed that the development of the standard is also closely related to the *Military Scenario Definition Language*. This is done by also mapping *Military Scenario Definition Language* to the 5 W's which makes it possible to further map it to a *Battle Management Language*. [9]

The *C-BML* study group report divides the development of the standard into three separate phases. After the completion of phase one, the standard should specify a data model for issuing military orders. The data model should be a subset of *Command and Control Information Exchange Data Model* using *XML*. Phase two will focus on developing a well documented and parse-able grammar for *C-BML*. The grammar will include syntax, vocabulary and semantics. Extensions to the grammar will be made so that reporting of military orders is possible. In the last phase, which is planned to be finished in April 2010, the standard should include a definition of the ontology of battle management. As a result, interoperability between services will be possible. [8]

1.3.7 geoBML

While other *Battle Management Language* solutions have focused on the domain and the structure of the language, *geoBML* addresses the surrounding environment in the planning of coalition operations. As forces move against a net-centric operation, more emphasis needs to be put on using the terrain and weather in the aid of *Coalition Command & Control*. The information must also be efficiently exchangeable between several coalitional parties. [1, 19]

1.3.8 Summary

The most interesting languages for use in *General Scenario Editor* are *Coalition Battle Management Language* and *Joint Battle Management Language* which are further investigated and evaluated in this thesis. This consideration is described in chapter 4.3.

2 Problem description

The purpose of this project is to extend the order management in the original *General Scenario Editor* to support *Battle Management Language* to the point where the result shows the possibilities of its concept. The user of the application should be able to assign military orders to different parts of the organization. This also involves exportation of a structured *Battle Management Language* from the scenario editor, and also exportation to a computer game's native scenario scripting format. The games of interest are *Steel Beasts 2 Pro* and *Virtual Battlespace 2*, both already having a working interface to the integration system *WISE Connectivity*. [6, 26, 27]

The academic aspect of this thesis is to find the core component that unifies the concept of *Battle Management Language*, and to decide which parts are the most beneficial for the intentions of *General Scenario Editor*.

The original *General Scenario Editor* only has very minor functionality concerning order management, which is only to the extent where it is possible to issue very specific tasks to the organization. These tasks can more accurately be seen as waypoints on the map. There is no way to export or import a structured *Battle Management Language* from and to the application. Nor is there a working function to export the orders to any of the applications used in *Saab Training Systems AB* current solutions.

The questions that will be answered in this thesis are the following:

- Which difficulties exist when designing a scenario editor with the support for order management and *Battle Management Language*?
- What data model will be needed to store the data in the scenario editor and how do we output this for exportation?
- Which extensions will need to be made to the scenario editor?
- Which orders could be simulated in the game?
- Which are the restrictions in the game?
- What must the orders look like in order to be imported into a specific game's format?

2.1 Visions and goals

On the market of military training there is currently considerable interest in *Battle Management Language*. To follow the development of this upcoming standard, *Saab Training Systems AB* wants to investigate how *Battle Management Language* can be integrated into their products, especially in the integration platform *WISE Connectivity*. In the end, *Battle Management Language* could be used in many aspects of military training, from multiple *Command & Control* systems to many different simulations and end systems.

The vision is also to combine the support for *Military Scenario Definition Language* with *Battle Management Language* in the scenario editor. This creates a complete solution for training scenario editing and is possible due to the organization structure of *Military Scenario Definition Language* and the military task assignment of a *Battle Management Language*.

The goal of this thesis is to demonstrate the functionality of supporting a *Battle Management Language* in the *General Scenario Editor* application. Practically, this is accomplished by going from a created scenario in the scenario editor, to a *Battle Management Language* and eventually to a specific game which simulates the scenario. The result can then be used to further investigate the use of *Battle Management Language*.

2.2 Scope

2.2.1 Design data storage for the extensions in *GSE*

As the original *General Scenario Editor* is very limited concerning order management, the initial focus is to extend the scenario editor to be able to store data in a way that makes it possible to store military tasks and output to a *Battle Management Language*.

The new data model must be able to:

- Store a military order, with all its essential data.
- Preferably use the data model already present for the *Military Scenario Definition Language* functionality.
- Be easy to export to a structured military order format.
- Be easily expandable for new purposes, additional order types and different *Battle Management Language* solutions.

2.2.2 Design and implement extensions to *GSE*

Functions to draw symbols and objects on a map are already implemented in the original *General Scenario Editor*. This implementation includes a subset of the common war fighting symbology defined in *MIL-STD-2525B*, which is a standard set by the United States military. As the original version of the scenario editor only includes symbols from *Appendix A* of the standard, it may be preferable to also implement support for the tactical oriented symbols from *Appendix B*. [12]

Depending on the solution, different features are needed to be implemented. However, some general features are required regardless of solution:

- A user interface for creating and assigning the military orders, using graphical- or text-input.
- Functions to represent military orders on the map and other preferred graphical extensions.
- Store information in the new data model.
- Retrieve all necessary information from the data model and export it to a *Battle Management Language*, possibly using some kind of export wizard.

2.2.3 Implement support for games

The material in the extended *General Scenario Editor* should be exportable into a game's native scripting language. The following steps are to be performed:

- Determine what possibilities the game
    - Analyze the game's scripting format.
    - Find the requirements, concerning input and output, for the game.
- If required, make additions to the different order types so that it can be successfully exported to a game.

2.3 Limitations

The extended *General Scenario Editor* serves as a proof of concept, and should not be considered a finished product for order management. With the extensions done during this thesis, it will be possible to further develop the scenario editor to a commercial product in the future.

The new, extended, data model will features the necessary parts for demonstrating the concept of order management. This model must be further extended in the future to support more order types and functions.

The military order types available in *General Scenario Editor* are "MOVE", "ATTACK" and "STRIKE" (see chapter 5.3.2). The only distinction between these order types when exporting to a game is that the units do not fire when performing a "MOVE"-order.

No additional symbols were designed as all needed could be retrieved from *MIL-STD-2525B.*

3 Work methods

3.1 Project model

Much of the work in this thesis is software development, which is performed using an appropriate project model. There is also some necessary research that is integrated with the software development work.

As the concept of a *Battle Management Language* is new to *Saab Training Systems AB*, the development is performed in small steps with constant evaluation of the progressing result. This makes it possible to detect serious mistakes and misconceptions at an early stage, and the work can be redirected without much lost work. It is also important for the company to be able to apply "trial and error" as a working method, where a function is implemented in a very simple way, only to show if it is reasonable or not. This is especially appropriate during development of user interface as it allows tryout and visualization.

The above description is similar to an evolutionary project model where there is always a working piece of software at the end of each cycle - the product "evolves". The advisor at *Saab Training Systems AB* weekly looks at the progress made and evaluates the result. After consultation with other advisors, a decision is made of how to proceed with the development and when to expect results. This approach is similar to the agile project model *SCRUM*, described on the official website of that project model, but without a defined set and lengths of sprints. [13] Such restrictions do not benefit the project, as the different application functions to implement vary in extent.

3.2 Working base structure

From the scope of this thesis, three obvious tasks can be distinguished:

- Data modelling
- Designing and extending *General Scenario Editor*
- Export to a game

Each task requires the gathering of information, which is performed in parallel with all tasks. This occurs intuitively as new information is revealed and assimilated during the development process.

The first and second tasks have the highest priority, and these are performed in parallel to some extent, as they are closely related. The third task has the lowest priority, which is also the reason why it is the last task and is performed as far as time allows it.

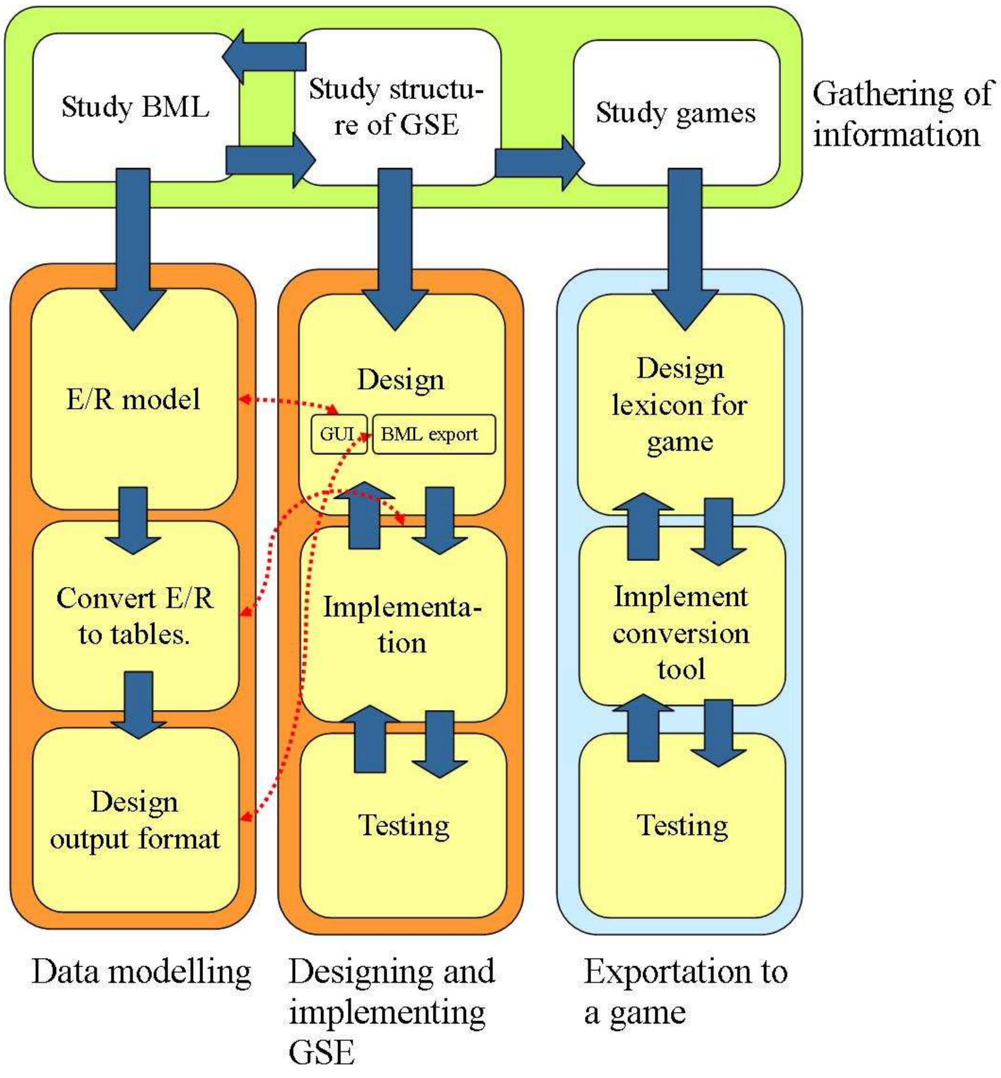


Figure 3-1: The project algorithm.

Figure 3-1 shows a graphical representation of the work structure. The vertical boxes can be seen as the three main tasks. The blue arrows represent the direction in which the work advances. The red dashed arrows show the relation between the main tasks. Such relation implies that those tasks are likely to be worked on in parallel with each other.

An observation that can be made in the horizontal box, is that the blue arrows are two-way for the first two subtasks but not for the last subtask. This implies that there should be no, or very little, modification to the work made in the previous tasks when reaching the "*Exportation to a game*" task.
The color of the vertical boxes indicates the priority of the tasks, where orange has higher priority than blue.

3.2.1 Gathering of information
As none of the participants of the project initially have any knowledge about *Battle Management Language* or much of its surrounding military terms, a lot of information is required and assimilated. This is done by reading a lot of material listed in chapter 7 and by consulting expert advisors. More information is revealed as the project advances and as advisors answer questions. This is taken in consideration when making decisions.

3.2.2 Task 1: Design output and data storage
When enough information is collected about *Battle Management Language* in general, a preliminary design of the data model and its content is created for *General Scenario Editor*. This data model contains most features needed for the proof of concept.

The second part of this task is to actually implement an export mechanism from the internal database of *General Scenario Editor* to a *Battle Management Language*.

3.2.3 Task 2: Design and create extensions to *GSE*
This second task is the most time consuming task and at its completion serves as the answer to the major questions of this thesis.

The main work in this task is to graphically design and implement the functions to the *General Scenario Editor*. During design, there is continuous communication with the advisor at *Saab Training Systems AB*.

The implementation includes the storage of the information input by the user into the defined database from the new data model. This is done by using existing functions for database management in the application. The result can later be exported to a *Battle Management Language*'s output format. The exact specification of this output format, along with the data model, is determined during Task 1.

3.2.4 Task 3: Implement support for exportation into games
Although a demonstration of the result is valuable, it is not vital for the proof of concept to demonstrate exportation to different games. This is the reason why this last phase has a lower priority.

There are two possible exportation methods that are considered depending on the result of the previous tasks.

- An export function integrated in the scenario editor that directly exports the material from the database to a game. In this way, all needed information is fetched directly from the database.
- Convert an exported *BML*-file independently of the scenario editor. An external conversion tool will need to be developed. It is however difficult

to produce any good results if there are too many limitations in the exported *BML*-file.

In this task, a “lexicon” is defined that translates the orders from the scenario editor in a way that the specific game can simulate it accordingly. A number of examples are developed to show some variations between order types in the game.

3.3 Design, Implementation and Testing

The first step is to design the data model of the database and the internal storage of *General Scenario Editor*. This is done after extensive research on *Battle Management Language* so that no serious mistakes are done at this point.

When a draft for the data model is done, it is possible to start the design for several different functions to be implemented in the scenario editor. Each function is processed one by one, which means that a function is designed, implemented and then tested before the next function is undertaken. Input from the advisor at *Saab Training Systems AB* and the other expert advisors will be considered during design and gives opportunity to experiment with different designs.

As the original *General Scenario Editor* is written in *C#*, the extensions made during this project are also done in the same language. *Saab Training Systems AB* provides some data structures, graphical components and methods that is usable. These include methods for interaction with the database, and the controller for drawing on and modifying the map.

Testing is integrated into the implementation process. No specific phase will be dedicated to just testing. All functions are first implemented as a draft and then presented to the advisor at *Saab Training Systems AB*. When the concept of the function is approved, errors from the draft implementation are corrected. After several functions have been implemented, approved and tested, approximately one complete work day is dedicated for finding and correcting errors in overall functionality in the application. After this, further implementation proceeds as usual. The goal is to ensure stable operation of the application without crashes induced by the extensions done during this thesis.

4 Proposed solutions

During the information gathering and design phase, different solutions were proposed and considered. In this chapter, the advantages and disadvantages is compared which leads to the result in chapter 5 and the conclusions drawn in chapter 6.

4.1 Focus

Due to limitations in time, compromises have to be made when deciding what focus the application design will have. There are two approaches to consider. The first choice is to concentrate on the representation of the order internally in the application, and to design the graphical functionality from there.
The second choice is to let a graphical representation of order management decide what the internal data structure will look like and thereby creating more visually appealing, and maybe more user-friendly, implementation.

4.1.1 Data structure focus

In this approach, the focus is put on the data structure for representing order within the application. A more thorough and flexible data structure makes it possible to expand the application more easily in the future. More time can also be spent on the exportation/conversion to the destined game (*Steelbeasts 2 Pro* or *Virtual Battlespace 2*). This gives a more elaborated demonstration of the whole chain of order management, all the way from the scenario editor to the games.

As less focus is put on the graphical representation of order management, it is possible that the result does not provide such a detailed view of the scenario just by looking at the map. Instead, simple symbols and colored lines represent orders. Some simple functionality with clickable symbols is also implemented.

A graphical interface that supports the underlying data structure is required. This interface is used to create and edit orders in a text-based manner. This interface can then easily be expanded to a more sophisticated graphical representation of orders in the future.

4.1.2 Graphical structure focus

If main focus is put on the graphical representation of the orders, this influences the data structure internally in the application. The internal data structure stores for example the placement of the symbols for the scenario, and maybe some additional information. During exportation to other formats, this graphical information is then interpreted and converted into textual order information.

This solution is of course preferable if the goal is to create a sophisticated graphical design regarding tactical orders. Complex symbols indicate specific orders and settings, which makes most of process of creating an order to be completely graphical. It is a matter of placing symbols on the map instead of textually defining the orders. Commercial products, such as *Military Overlay Editor* (MOLE) [23], are considered as a potential candidate to use as graphical user interface for this solution.

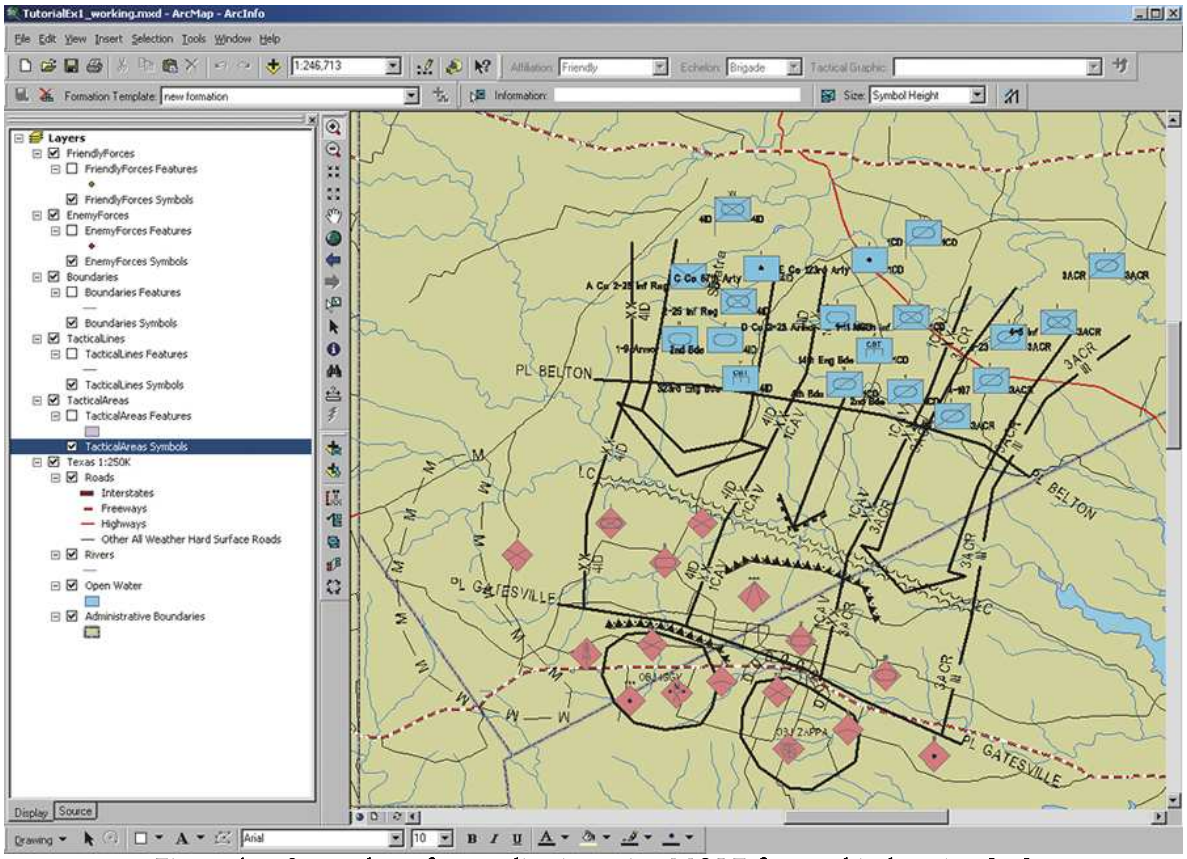


Figure 4-1: Screenshot of an application using MOLE for graphical tactics. [23]

From a product point of view, the profit of making an advanced graphical representation of orders is that the promotion of the product is potentially easier. The problem of this concept is however the difficulty in both design and implementation of such a user interface. Interpreting graphical orders could not only lead to ambiguousness but also introduce limitations for order management. If such limitation is found late in the development process, it is difficult to redesign and change. The outcome of this solution is thereby very uncertain and is highly dependable on time. This because serious limitations can be encountered late during development, especially when trying to export orders to different output formats.

Comparing advantages and disadvantages, “Data structure focus” is the most advantageous option and is used in the implementation.

4.2 Orders structure in *General Scenario Editor*

The goal is to make extensions to the scenario editor without interfering with the existing structure of the original version. Two solutions are proposed concerning the internal order structure in *General Scenario Editor*, and the differences are discussed in this chapter.

4.2.1 Fixed order structure

To be able to start implementing the new extensions quickly into the scenario editor, a fixed structure for each order type (verb) is created. This is less time consuming as it is only necessary to consider the order types that this thesis intend to implement, with not much concern of future extensions. The data model is also easy to design.

This approach may be tempting in order to see results early in the implementation process and to reach the goals of the thesis in time. It is however an irresponsible solution regarding future development of the scenario editor concerning order management.

4.2.2 Generic order structure

The fact that many orders share the same structure can be exploited to create a flexible internal data structure for managing orders. For this, it is necessary to study *Command & Control* lexical grammar and *Battle Management Language* to identify common parameters and their possible values. If these are correctly obtained, a universal order base can be created with all of these common parameters. This simplifies the process of extending the scenario editor with new order action types in the future. Even the external database is arranged in such a way that this “flexible” property is supported.

The disadvantage of this solution is that it can become very complex and cause some overhead. From the scope of this thesis, this solution is not reasonable because of the small quantity of order types implemented. It however opens up the possibility for additional order types and is therefore the chosen solution.

4.3 *BML* used in the implementation

There are different *Battle Management Languages* defined, all of them have some similar characteristics. It is however possible that some of these languages are more appropriate to use in the *General Scenario Editor* than others. Looking at the table in chapter 1.3.2, the two languages *Coalition Battle Management Language* and *Joint Battle Management Language* are chosen as candidates for the implementation. These two languages are examined further and which of them to implement is decided.

4.3.1 Coalition Battle Management Language

Coalition Battle Management Language is a standard in progress and is considered new and fresh. The development of the standard is divided into three phases. This phase consists of developing a formalized and structured language to be transferred to the 5 W's principle.

There are draft data models that can be used for implementation in *General Scenario Editor*. The drawback of using the draft version of *Coalition Battle Management Language* in this thesis is that it can be hard to find information that can be used. The draft *XML* schema definition files may also be lacking some functionality.

4.3.2 Joint Battle Management Language

As mentioned in previous chapter, *Joint Battle Management Language* is developed for the United States military and contains both a specification and an implementation. There is a working data model available and it is easier to find substantial information about it, which is the main reason why *JBML* is chosen as the language to implement.

The differences between the candidate languages do not affect the result significantly. As Andreas Tolk explains in one of his papers, the work made during the development of *Joint Battle Management Language* is also the main contributor to the development of the current *Coalition Battle Management Language*. [3] So even though *JBML* is quite old itself, the concept and features of the language is not obsolete. This means that extensions made to the *General Scenario Editor* is not in vein no matter which of the two languages are selected. The purpose of this thesis is to demonstrate the proof of principle with *Battle Management Language* in a scenario editor. As the extended *General Scenario Editor* stores order information in a generic manner it is not very difficult to also export to other languages as well.

4.4 Building appropriate *BML* output

Users of the application likely have different preferences about the output format. It is necessary to take this into consideration when deciding a solution.

4.4.1 Output format

Joint Battle Management Language provides schema definition files for *XML* output, which can be used to structure an order. The schema definition files will make it very convenient to export data from the data model to a *XML*-file containing the military orders. This is the most logical approach since *General Scenario Editor* already includes functions to construct *XML*-documents.

It is of course possible to study the schema definition files and create a proprietary format for exportation. This is however redundant work as *XML* libraries are already available.

4.4.2 Combine *Military Scenario Definition Language* and *BML* output

Saab Training Systems AB looks at the possibility to combine the *Military Scenario Definition Language* and *Battle Management Language* functionalities in the scenario editor. This also opens a discussion if the exported output should combine both of these formats into one output format.

The original *General Scenario Editor* supports the functionality to export the organization of a military scenario to *Military Scenario Definition Language 1.0*. *Joint Battle Management Language* also supports the representation of the units and resources involved in the order using its own modified version of *Military Scenario Definition Language*. So the question is whether to use this modified *Military Scenario Definition Language* version or to deviate from *Joint Battle Management Language's* definition and instead use *Military Scenario Definition Language 1.0*. By doing so, the original schema definition files for *Joint Battle Management Language* cannot be used when validating the output.

In some cases it is also desirable to completely separate the organization and the order management, creating separate output files for the *Military Scenario Definition Language* and *Battle Management Language*.

The decision is to implement all of these variations, and introduce them in an export wizard in the application. Then the user can decide what output he or she wants. This approach will of course be more time consuming than to just implement one variation, but it will make it possible to evaluate different aspects of *Battle Management Language* and *Military Scenario Definition Language*.

4.5 Exporting to a game

As stated in the problem description (chapter 2) there are two games of interest concerning the exportation from the extended *General Scenario Editor.* These games are *Steel Beasts 2 Pro* and *Virtual Battlespace 2.* The investigation revealed that the mission files for *Steel Beast 2 Pro* was binary formatted and undocumented. To a developer, the difficulties in that situation are obvious and therefore *Virtual Battlespace 2* can be determined as the game of choice. This game uses a text-based and scriptable mission format, yet undocumented.

Although *Virtual Battlespace 2* has a lot of possible settings, these specific properties can be set to a default values for all scenarios. This makes the 5 W's principle used in the *General Scenario Editor* sufficient for structuring a basic order even in the game. Any specific vital information is either configured in an export wizard, or using optional input during the order creation.

4.5.1 What scenario could be shown in the game?

The investigation of the game and its mission editor revealed that it is possible to simulate both simple and quite advanced orders within the game.

4.5.1.1 The basic order

The orders in *Virtual Battlespace 2* (VBS2) are, in the most basic form, just waypoints that can be assigned to the units present in the scenario. All units need to be a member of a group. A waypoints assigned to a group leader applies to all of the group's members. No orders for individual units of a group are allowed. All waypoints can be configured independently to affect the approach of the units using the waypoint. The primary option for a waypoint is its "Type", which is a setting that affects the overall execution of the waypoint. A more detailed description of these types can be found in the *VBS2 Editor Manual.* Other settings include the units' behavior in combat situations and the speed of movement. [22]

4.5.1.2 The more advanced order

Virtual Battlespace 2 has a built-in scripting format that allows the user to create very complex missions. Script code snippets can be inserted for most objects in the scenario. Also a specific object, called "Trigger", can be placed on the map to allow script code to be run when a specific condition is valuated true.

As an example, the condition could be set to: "Activate trigger if a vehicle enters a 100 meter radius of the trigger's position". The activation of a trigger basically means that its "activation statement" is executed. Such statement could for example be: "Reduce the activating vehicle's fuel tank to half".

There are many functions in the scripting language that allows very detailed coordination of the units' behavior in the scenarios. This opens up the possibility to make AI units appear making decisions, just by using a set of well placed triggers.

With the scope and timeframe of the project in mind, it is decided to only aim for some basic order translation from the scenario editor to the game. This would not require extensive scripting.
With a simple "Move"-waypoint, both order types "MOVE" and "ATTACK" from the scenario editor can be translated to the game. To distinguish the two order types in the game, the waypoints used for "MOVE" have a "Hold fire"-setting applied, while the "ATTACK" have a "Fire at will" setting applied. This somewhat naive distinction is enough to at least prove the concept.

5 Result

The result of this thesis can be seen as the new functions to the scenario editor. For a more technical explanation of some functions refer to appendix in chapter 9.

5.1 Mapping to the 5 W's

The first thing that had to be done was to decide what input that should be available to the user. As *Joint Battle Management Language* relies on the 5 W's principle, this was a good approach when finding the different properties of the order that could be input. To be able to build a relevant order, some information is crucial while some can be optional. When designing the database and the graphical user interface all these properties were considered and are expected to work even if the solution is extended to support other output formats in the future. [3, 17]

5.1.1 What

"What" defines the verb of the order, which tells what kind of action that is to be executed. It is important that the sender and the receiver of a military order share the same lexicon so that an order can not be misinterpreted. The extensions made to *General Scenario Editor* resulted to include only three verbs.
The specific "What" may also affect how the related W's and additional information will be structured.

5.1.2 Who

Depending on the chosen verb in the "What" and the *BML* used, different types of "Who" can exist in an order. In every *BML* encountered during this thesis at least two types of "Who" need to be specified in each order -"Tasker" (issuer) and "Taskee" (performer). However, additional "Who"-parameters, such as "Affected", can be added depending on the situation.

5.1.3 Where

As every military order need to have some kind of reference concerning where it should be executed, this parameter must be very flexible for adding information about different types of locations. Sometimes the information has to be more describing than just a simple coordinate. For example, "Move along the river" or "Approach unit X at a radius of 200 meters". It might also be of interest to have information regarding "From Where". This information can be used to execute more complex orders, such as "Ambush".
In *Joint Battle Management Language*, it is possible to specify what type of *category* (Route, Minefield etc) and the *classification* (Point, Area or Line) the location is. Also a *qualifier* should be chosen, which states how the approach to the location should be performed (Along, Between, To etc).

5.1.4 When

Basically there could be two properties of "When" in an order. One which tells when the order should be executed and there could also be one which tells when an order should end.

In most *Battle Management Language* solutions there are also two types of When-parameters. One that specifies a time and another that specifies a reference to another order. There could also be a qualifier ("Not before", "At time", "ASAP" etc) with which it possible to state for example that an order should not be executed before another order has been completed.

5.1.5 Why

This part of an order could basically be a text-string that describes the purpose of the order. It can also be structured in a formal language which is the case with *Joint Battle Management Language*. An example of a Why-parameter would be "in-order-to surprise". In *Joint Battle Management Language*, "Why" is an optional parameter and is not implemented in *General Scenario Editor*.

5.1.6 Other information

Depending on the *Battle Management Language*, there can be additional information supplied with an order. For example, the *Joint Battle Management Language* offers the possibility to choose a transport-type for ground positioned units, while some other *Battle Management Language* solutions supports the specification of a formation of the units included in an order.

5.2 Data model

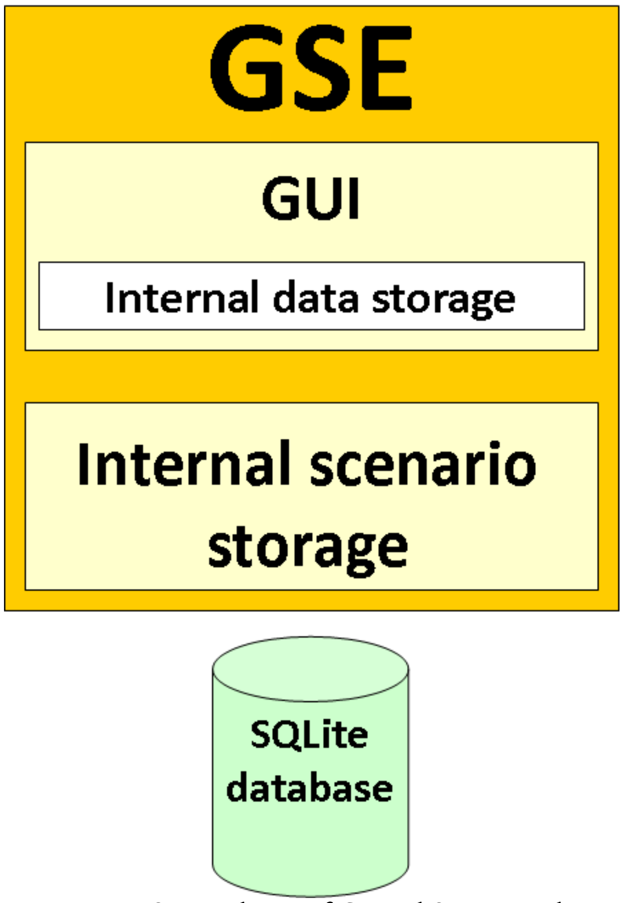


Figure 5-1: Storage layers of General Scenario Editor

It is possible to distinguish three layers regarding data storage in *General Scenario Editor*. The top layer consists of the graphical user interface (GUI), which presents information to the user and selects the information that should be forwarded further down.
Before the information is passed down to the next layer it passes "Internal data storage" of the GUI. This layer synchronizes the new information that is sent down, so that the same information also is presented in the GUI. This especially applies to the tree-hierarchy view that always should present a correct view of the scenario's organization.

The information travels down to the next level, "Internal scenario storage". This layer stores a copy, as a local cache, of the whole external database – a *SQLite* database. The interface to above layers consists of methods for fetching and altering the local cache. Eventually the local cache and the external database must be synchronized, which is done when the user decides to save the scenario.

The main task of the development has been to design the internal data storage of the scenario editor to support order management. There are basically three parts that has been designed:

- Data structures for internally representing order information
- Internal storage of information
- An external database (tables) that supports the internal data structure

5.2.1 External database

As the extended *General Scenario Editor* does not feature any extensive set of order types, it is required that the database is easy to expand in the future. A lot of work has been done in making a generic data model internally in *General Scenario Editor* that supports the external database.

5.2.1.1 E/R model

As seen in the picture of Appendix 9.2, an object oriented view is present even in the database, where the different order types share a common base of properties. It should be pointed out that there is information stored in this data model that is not used by *Joint Battle Management Language*. This is because the goal was not to create a "*Joint Battle Management Language* editor" but a generic scenario editor that can be used for exporting to different formats. This will also be advantageous when extending the export possibilities to different *Battle Management Language* formats in the future.

5.2.1.2 SQL tables

The different entities from the E/R model is implemented as separate tables instead of just combining them into one. In this implementation, where there are only two classes of orders, this solution might not seem appropriate. But when the number of order types increases, a separation of the tables will probably be very advantageous as different parameters (fields) may apply to different order types.

Table: OrderBase

This table contains the common properties of an order. This can be directly linked to the class *BMLActionType* in the internal data structure of the scenario editor.
The *actionType* field in *OrderBase* determines which additional table that is necessary to complement the order.

Field	Description
orderId	Unique id for an order.
actionType	Name of the order type.
orderName	Name of the order. Should be unique.
scenarioId	Specifies which scenario this order belongs to. Refers to the table “Scenario”.
taskerType	Type of tasker. Can only be “Unit” in the current implementation.
taskerId	Id of the type of tasker specified in *taskerType*.
taskeeType	Type of taskee. Can only be “Unit” in the current implementation.
taskeeId	Id of the type of taskee specified in *taskeeType*.
why	A textual representation of the intent of the order. Not implemented at the moment.
startConditionType	Type of the start condition. Can be “Time condition” and “Action condition”.
startConditionQualifier	Qualifier of the start condition.
startConditionAction	Id of another order, specified for an “Action condition”.
startConditionTime	Time for the condition, specified for a “Time condition”.
endConditionType	Type of the end condition. Can be “Time condition” and “Action condition”.
endConditionQualifier	Qualifier of the end condition.
endConditionAction	Id of another order, specified for an “Action condition”.
endConditionTime	Time for the condition, specified for a “Time condition”.
WhereType	Type of location. Can be “Coordinates”, “Route” and “Indirect unit”.
WhereCategory	Category of location.
indirectUnit	Id of another unit, specified for “Indirect unit”.
routeId	Id of a route, specified for “Route”.
Longitude	Longitude, in degrees.
Latitude	Latitude, in degrees.

Table: OrderMove

Orders that can be classified as movement orders should be placed in this table. In the internal structure of the scenario editor, this corresponds to the subclass *BMLMoveActionType*. In the implementation this table is used for the order type “Move”.

Field	Description
orderId	Unique id for an order. References an order id from the table *OrderBase*.
Speed	Speed associated with the move order.
Formation	Formation associated with the move order.

Table: OrderAttack

Orders that involve attacking a target should be placed in this table. In the internal structure of the scenario editor, this corresponds to the subclass *BMLAttackActionType*. In the implementation this table is used for the order types "Attack" and "Strike".

Field	Description
orderId	Unique id for an order. References an order id from the table *OrderBase*.
Resource	A resource to be used when executing the attack order.

Table: AreaOfInterest

With this table, it is possible to globally define a collection of coordinates for the scenario. In this implementation of the scenario editor, only "routes" can be defined. The idea is that "areas" and "single points" should be supported in the future.

Field	Description
aoiId	Unique id for the collection of coordinates.
scenarioId	Specifies which scenario this order belongs to. Refers to the table "Scenario".
name	Name for the collection of coordinates.
type	The type of collection. Can only be "ROUTE" in the current solution.
coordinates	A serialized text string of an array with coordinates.

5.2.2 Classes

The classes for structuring orders in *General Scenario Editor* are divided into three data structures.

- *BMLActionType*, which contains almost all information about the task.
- *BMLCondition*, which is used for specifying some kind of condition for the order.
- *BMLOrder*, mainly consists of the two types above.

As the original version of *General Scenario Editor* implemented simple functions for handling orders and actions, many obvious class-names were taken. This is the reason why the keyword "*BML*" was added for most new classes. This is actually not implying that the use of these classes are restricted for use with *Battle Management Language*, it is only done to not cause any confusion.

5.2.2.1 BMLOrder

When all information is gathered, it is packed into a final *BMLOrder* instance. The main member attribute of this class is an instance of *BMLActionType*. Other information included is an identification string and the name of the order. There are also two *BMLCondition* objects attached; one for specifying a start condition and another for the optional end condition.

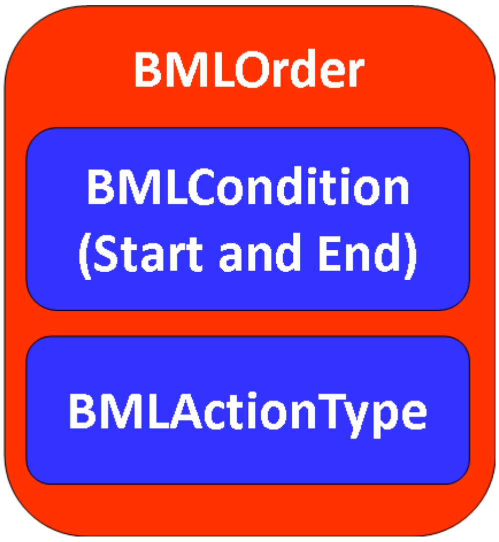


Figure 5-2: Content of the BMLOrder-class

5.2.2.2 BMLActionType

The biggest component of an order is the *BMLActionType*. This structure basically contains all information given in the order editor about the military task. It also deals with the interface to the property grid (see terminology) in the order editor. More specifically, it generates and synchronizes its attributes with the options available in the property grid of the order editor window.

The *BMLActionType* contains all information considered as fundamental for an order, based on the 5 W's principle. This makes it possible to extend this class with more functionality to include many other, more specific, order types. Two of these are created for demonstration:

- *BMLAttackActionType* – see respective chapter 5.3.2.1
- *BMLMoveActionType* – see respective chapter 5.3.2.2

Each of these subclasses also has their own table in the database to demonstrate how further extensions of order types could be separated from each other. A corresponding solution would instead be to include all the new fields in the *OrderBase*-table.

5.2.2.3 BMLCondition

An order always requires some kind of condition for it to begin, whether it is "as soon as received" or a specific time. It could also be a condition connected to another event. To assemble this concept, *BMLCondition* is designed as a basic structure to be attached onto a *BMLOrder*. The structure can store a time, a reference to another order and a qualifier.

The information stored in *BMLCondition* is also accessible from the *BMLActionType* attributes. It may therefore seem redundant to include

BMLCondition in the *BMLOrder* as well. The argument for this solution is to further emphasize the distinction of conditions and the actions that should be performed in the task.

5.3 *General Scenario Editor* functions

5.3.1 Routes

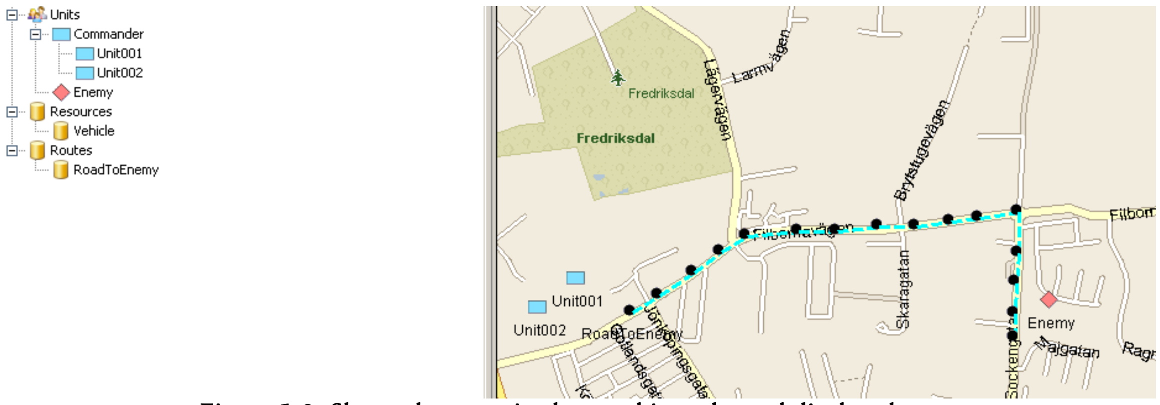


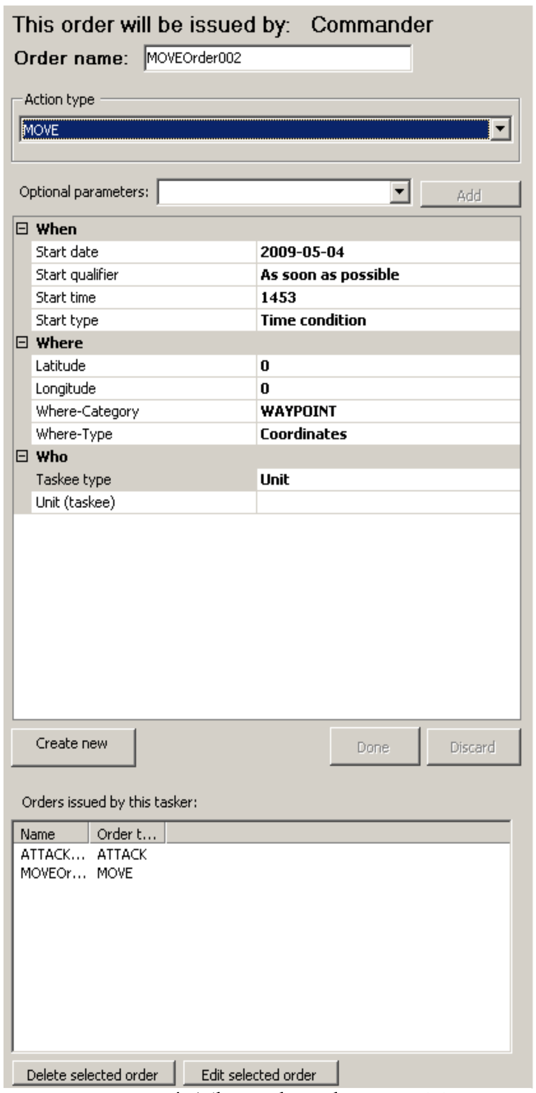
Figure 5-3: Shows the route in the tree-hierarchy and displayed on map.

The original *General Scenario Editor* lacked the possibility to draw routes and other areas of interest on the map. Such areas of interest can be very useful when issuing an order. It was decided to extend the functionality to include routes in *General Scenario Editor*. As the same route could possibly be used for many different orders, routes are made global to the entire scenario. The route-editing functionality is put in the tree-hierarchy (see Figure 5-3). With a simple click, the user can proceed to draw a route and give it a name. This name is then used to assign the route as the destination for an order. As the routes are global, they are also put in their own table in the database, see 5.2.1.2.

Internally, the routes have a header class named *Route*. The header, basically a header for a linked list, references the first and last waypoint of the route. The waypoints within belongs to the class *Waypoint*. The waypoints themselves keep track of the following and previous waypoint. The coordinates of these waypoints are serialized to the database to fit in a single table row.

5.3.2 Order editor

The order editor is the tool for creating and editing orders in the scenario and is the main functional extension made to *General Scenario Editor*. The editor utilizes the existing organization management in the scenario editor for issuing orders between them.


This order will be issued by: Commander

Order name: MOVEOrder002

Action type: MOVE

Optional parameters:

When	
Start date	2009-05-04
Start qualifier	As soon as possible
Start time	1453
Start type	Time condition
Where	
Latitude	0
Longitude	0
Where-Category	WAYPOINT
Where-Type	Coordinates
Who	
Taskee type	Unit
Unit (taskee)	

Orders issued by this tasker:

Name	Order t...
ATTACK...	ATTACK
MOVEOr...	MOVE

Figure 5-4: The order editor in GSE

The intent was to make a simple graphical interface to the internal data storage of the scenario editor. All settings of an order are possible in the order editor window. This may not be very user friendly alone, but when extending the window to interact with the map and other means of input, the concept is actually very user-friendly and useful.

To use the order editor, one simply selects the unit in the tree-hierarchy and this choice will be set as the tasker (issuer) for the order. In the order editor, an order type is selected and its available properties are presented in the property grid. A name of the order can be specified during the creation of the order, and all other settings are done by changing the properties in the property grid.

Previously created orders can be loaded and edited, or deleted, from the order list which is presented at the bottom of the window. This order list presents the created orders from the currently selected tasker.

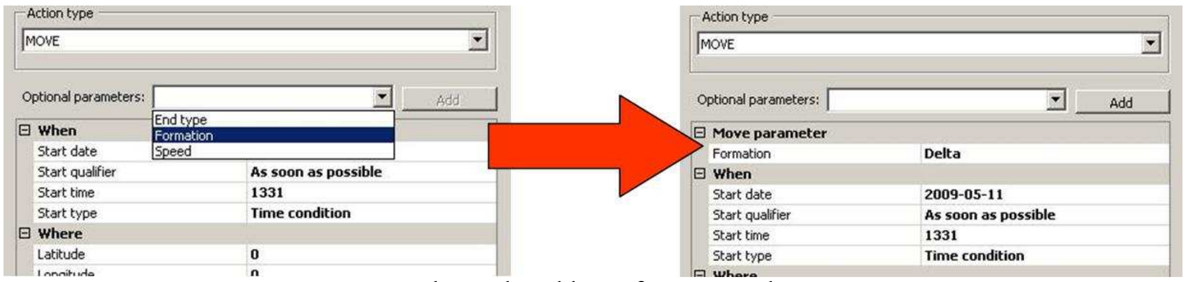


Figure 5-5: Shows the adding of an optional parameter.

Another feature concerning orders, are the ability to add optional properties to the order. This enables the user to increase the detail of the order when the situation requires it.

5.3.2.1 Attack and Strike order

In the military doctrine, there are differences between the meaning of “Attack” and “Strike”. “Attack” is a general attack while “Strike” is more aggressive and should result in the total defeat of the target and the control of its position. In the scenario editor however, the only difference between them is the name. This similarity allows both to share the same class in the internal structure of the application. Only one special property differ the "common order" from these order types, which is the ability to specify a resource (such as a weapon) to be used during execution of this order.

5.3.2.2 Move order

The "Move" order type was implemented due to the simplicity of the order, as it only requires an end point in the form of end coordinates. Also some optional parameters where added, like "Formation" and "Speed", to show the flexibility of the order creation.

5.3.2.3 Where-parameter

The options for defining the "Where"-parameter were constructed with the support for the *XML* schema definition files in mind. This was the most logical solution, as it would be easy during exportation to *Joint Battle Management Language*. [17]

The schema definition files offered three different ways of defining the "Where"-parameter:

- To a coordinate
- To an indirect unit
- To a set of coordinates

This introduced three options in the order editor:

- "Coordinate", which refers a coordinate on the scenario map
- "Indirect unit", which refers the position to a unit in the organization.
- "Route", which refers to a previously defined set of waypoints

5.3.2.4 When-parameter

The tasks of a military operation have a start-condition and optionally an end-condition. These conditions are specified in two ways according to *Joint Battle Management Language*: [17]

- By a point in time
- As a relation to another task

This also introduced two types of options in the order editor:

- "Time condition", which is set to a date and time
- "Action condition", which is set to another order present in the scenario.

In both cases, it is also possible to set a qualifier for the condition, for example "After" and "As soon as possible".

5.3.2.5 Interaction with the map

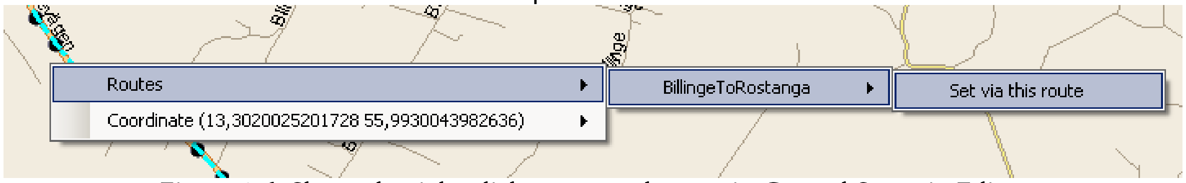


Figure 5-6: Shows the right-click menu on the map in General Scenario Editor

During the process of creating an order, the user is able to right-click on the map to list some of the objects in that area. Such objects can be units, routes, coordinates and other existing orders. Moving deeper in the menu-tree, the user can select options that reflect choices in the order editor. For example one can set a specific unit on the map as the target of an attack by right-clicking nearby and click "Set as target" for that unit.

The right-click menu features the following list of options for different objects on the map:

- For a Unit
  - Set as target for the Where-parameter.
  - Set as the performer of the order.
- For a Route
  - Set order to be executed via route.
- For an other order
  - Set as start-condition for the order.
  - Set as end-condition for the order.
  - Modify… (opens the order in the order editor)
- For the clicked coordinate
  - Set as target for the Where-parameter.

5.3.3 *Order view*

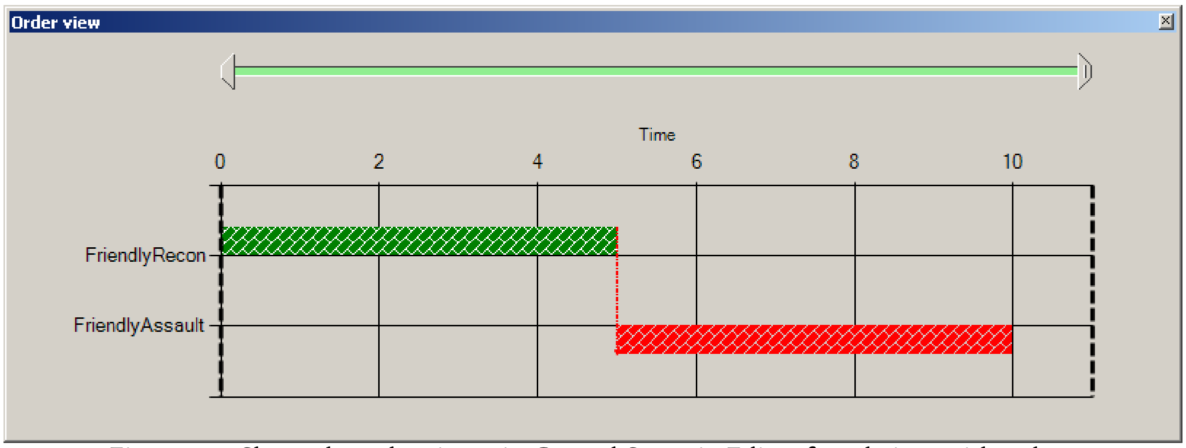


Figure 5-7: Shows the order viewer in General Scenario Editor for relations with orders.

By default, when a unit is selected in the tree-hierarchy, all orders issued by this unit are presented on the map. This unit is usually called the *tasker*.
The *order view panel* gives the user two useful functions for controlling the way orders are visualized in the scenario.

- It visualizes the whole scenario for the selected tasker in a chart. In this chart, orders are grouped by taskee (performing unit) vertically and placed horizontally regarding time. By clicking on an order in the chart, that order is editable in the order editor window.
- By using the range bar (time line) above the chart, the user may even further decide what orders should be visible on the map. This avoids the map from being cluttered with too many orders.

5.3.4 Export to *Joint Battle Management Language*

In order to show the result of the order management in the extended *General Scenario Editor*, the content of a scenario is exportable to *Joint Battle Management Language* (JBML).

5.3.4.1 Processing the data for exportation

XML schema definition files from *JBML 1.4* are used as the format for output. This makes it fairly easy to export the data stored in the database.

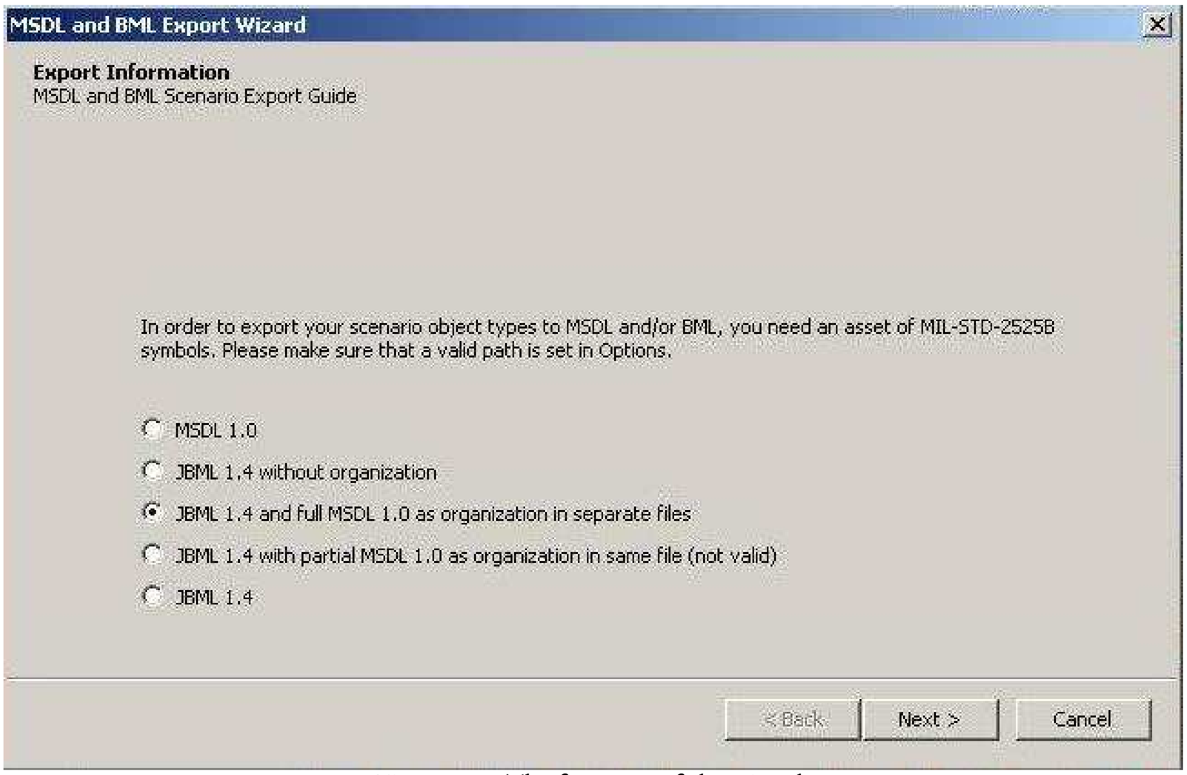
Although the exportation overall is quite straightforward, some fields in the schema definition files can not be determined easily from the internal database. In this case, they were left to a default value.

Even if *Battle Management Language* and *Military Scenario Definition Language* are closely related, the new export handler does not interfere with the original *General Scenario Editor* export mechanism for *Military Scenario Definition Language*. Instead, the new export handler for *Joint Battle Management Language* is called within the old export handler when needed.

For further details about the internal procedure of exportation, see Appendix 9.5.

5.3.4.2 Export wizard

An export wizard is implemented to make the exportation to *Joint Battle Management Language* as user-friendly as possible. The export wizard is actually an extension of the previous *Military Scenario Definition Language* export wizard.


MSDL and BML Export Wizard

Export Information
MSDL and BML Scenario Export Guide

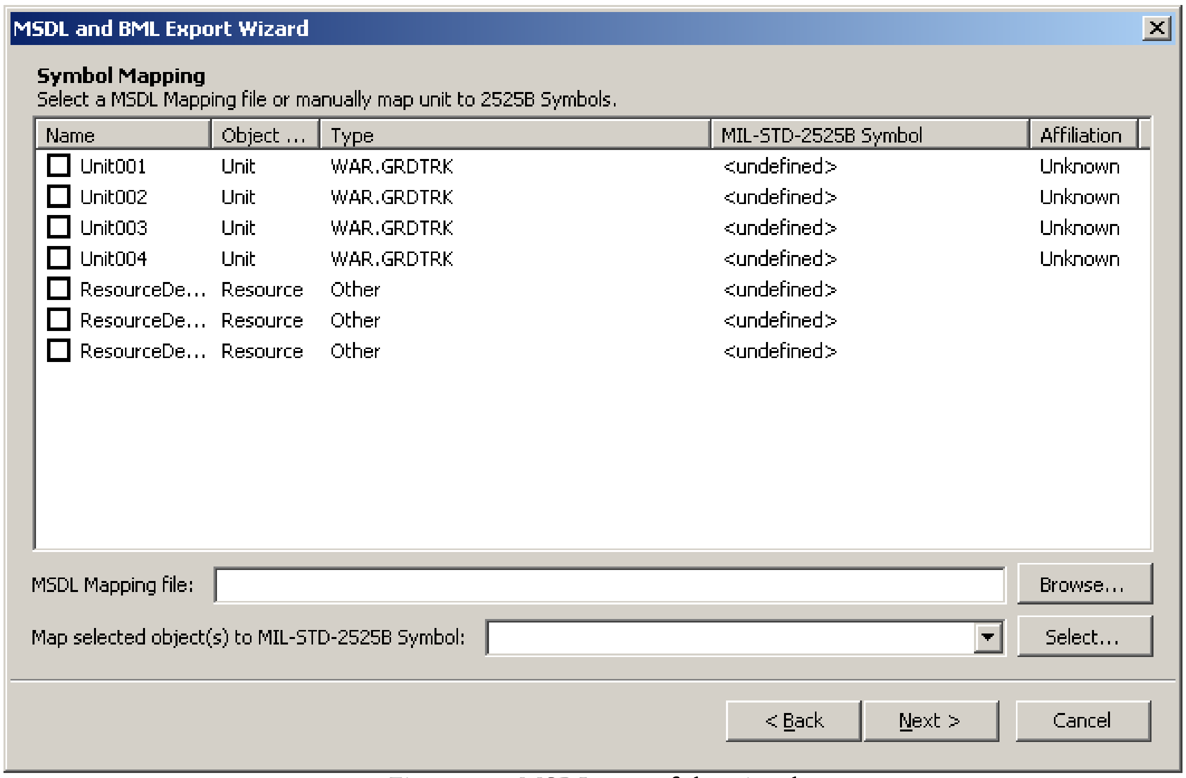
In order to export your scenario object types to MSDL and/or BML, you need an asset of MIL-STD-2525B symbols. Please make sure that a valid path is set in Options.

- MSDL 1.0
- JBML 1.4 without organization
- [✓ JBML 1.4 and full MSDL 1.0 as organization in separate files]
- JBML 1.4 with partial MSDL 1.0 as organization in same file (not valid)
- JBML 1.4

Figure 5-8: The first step of the wizard.

The export wizard makes it possible for the user to export a scenario to *Military Scenario Definition Language* and *Joint Battle Management Language* in five different output-modes:

- *Military Scenario Definition Language 1.0* only
- JBML 1.4 with organization part removed.
- JBML 1.4 with *Military Scenario Definition Language 1.0* as the organization part, in separate output files.
- JBML 1.4 with *Military Scenario Definition Language 1.0* as the organization part, in the same output file.
- JBML 1.4


MSDL and BML Export Wizard

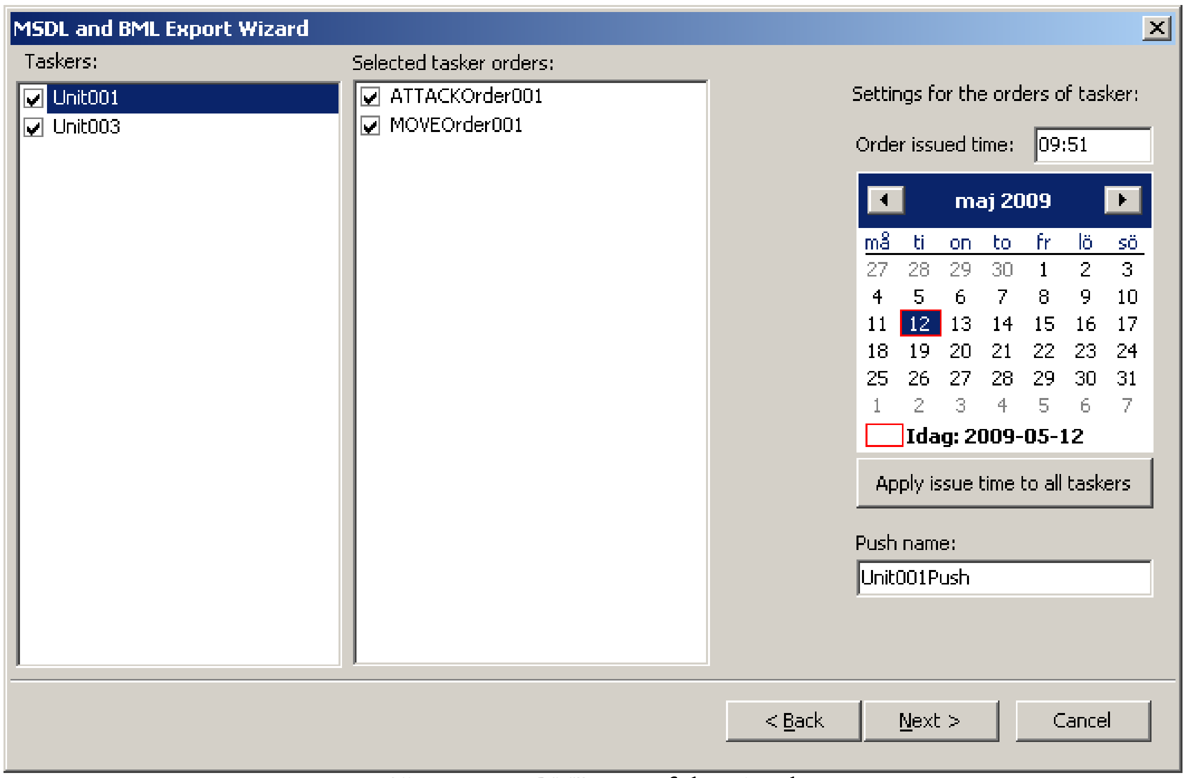
Symbol Mapping
Select a MSDL Mapping file or manually map unit to 2525B Symbols.

Name	Object ...	Type	MIL-STD-2525B Symbol	Affiliation
☐ Unit001	Unit	WAR.GRDTRK	<undefined>	Unknown
☐ Unit002	Unit	WAR.GRDTRK	<undefined>	Unknown
☐ Unit003	Unit	WAR.GRDTRK	<undefined>	Unknown
☐ Unit004	Unit	WAR.GRDTRK	<undefined>	Unknown
☐ ResourceDe...	Resource	Other	<undefined>	
☐ ResourceDe...	Resource	Other	<undefined>	
☐ ResourceDe...	Resource	Other	<undefined>	

MSDL Mapping file:

Map selected object(s) to MIL-STD-2525B Symbol:

Figure 5-9: MSDL-step of the wizard


MSDL and BML Export Wizard

Taskers:
☑ Unit001
☑ Unit003

Selected tasker orders:
☑ ATTACKOrder001
☑ MOVEOrder001

Settings for the orders of tasker:

Order issued time: 09:51

maj 2009

må	ti	on	to	fr	lö	sö
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

Idag: 2009-05-12

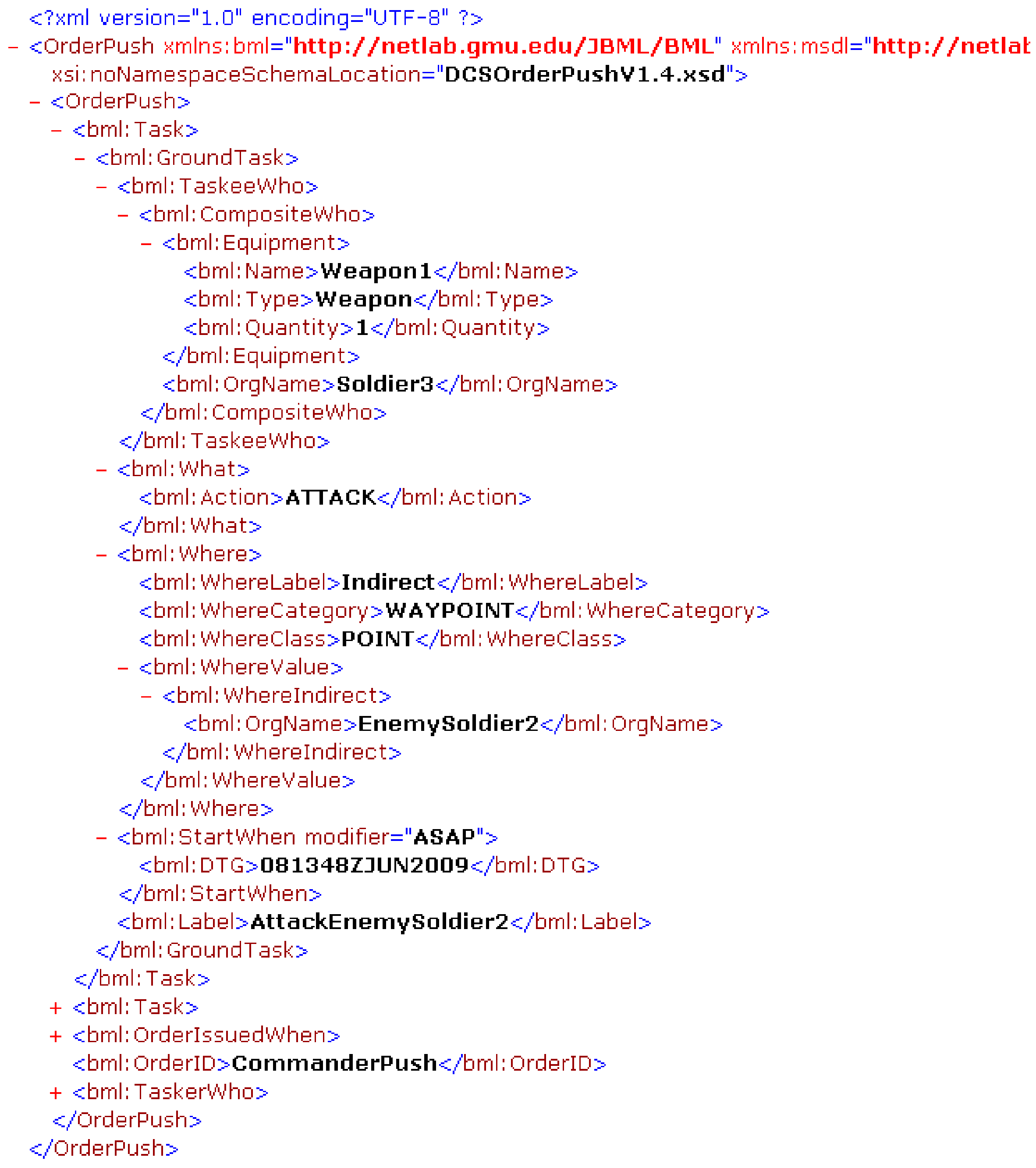
Push name: Unit001Push

Figure 5-10: BML-step of the wizard

According to *Joint Battle Management Language*, each order created with the scenario editor is exported into a “task” in the outputted *XML* file. Every task has

to be a part of an “order push”, which is simply a collection of tasks from the same issuer (tasker) at a given time. The export wizard makes it possible for the user to select which issuer’s orders to export and outputs these “order pushes” in separate files. The user can also choose to exclude a specific order.
For each issuer, it is also possible to specify settings for the order push, such as issuing time and an appropriate name for the push.

An example of *Joint Battle Management Language* output from *General Scenario Editor* can be observed below:



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <OrderPush xmlns:bml="http://netlab.gmu.edu/JBML/BML" xmlns:msdl="http://netlat
    xsi:noNamespaceSchemaLocation="DCSOrderPushV1.4.xsd">
  - <OrderPush>
    - <bml:Task>
      - <bml:GroundTask>
        - <bml:TaskeeWho>
          - <bml:CompositeWho>
            - <bml:Equipment>
                <bml:Name>Weapon1</bml:Name>
                <bml:Type>Weapon</bml:Type>
                <bml:Quantity>1</bml:Quantity>
              </bml:Equipment>
              <bml:OrgName>Soldier3</bml:OrgName>
            </bml:CompositeWho>
          </bml:TaskeeWho>
        - <bml:What>
            <bml:Action>ATTACK</bml:Action>
          </bml:What>
        - <bml:Where>
            <bml:WhereLabel>Indirect</bml:WhereLabel>
            <bml:WhereCategory>WAYPOINT</bml:WhereCategory>
            <bml:WhereClass>POINT</bml:WhereClass>
          - <bml:WhereValue>
            - <bml:WhereIndirect>
                <bml:OrgName>EnemySoldier2</bml:OrgName>
              </bml:WhereIndirect>
            </bml:WhereValue>
          </bml:Where>
        - <bml:StartWhen modifier="ASAP">
            <bml:DTG>081348ZJUN2009</bml:DTG>
          </bml:StartWhen>
          <bml:Label>AttackEnemySoldier2</bml:Label>
        </bml:GroundTask>
      </bml:Task>
    + <bml:Task>
    + <bml:OrderIssuedWhen>
      <bml:OrderID>CommanderPush</bml:OrderID>
    + <bml:TaskerWho>
    </OrderPush>
  </OrderPush>
```

5.3.4.3 Validation of the output

Using the supplied *XML* schema definition files and the built-in functionality in C#, it is possible to validate the output from the scenario editor. This validation basically means that the content and the structure of all elements in the *XML* document are checked to ensure that everything is as expected.

The supplied schema definition files cannot be used if the material to validate is *Joint Battle Management Language* using *Military Scenario Definition Language 1.0*. This is due to the collision of an identical namespace name ("msdl:") between the schema definition files.

5.4 Export to Virtual Battlespace 2

The result of the exportation to *Virtual Battlespace 2* demonstrates a very basic order execution. Although it is possible to make orders very complex in *Virtual Battlespace 2*, the only distinction between "MOVE" and "ATTACK" was "move, hold fire" and "move, fire at will". This is explained in previous chapter 4.5.

5.4.1 Output format

A *Virtual Battle Space 2* mission consists of three different files. These files are stored in a folder according to the pattern *<TheMissionName>.<ShortMapName>*, for example *ScenarioTest.Sama*. [22]

mission.sqm
This file is the first to be read by the game. It contains information about the initial point of the scenario, which includes the definition of the participating units and general settings for the mission. No information regarding the actions of these units exists in this file. [22]

mission.sqf
This is the script file which executes after *mission.sqm* has been loaded. This will activate all the actions of the scenario. The file is generated using the *mission.biedi* file. [22]

mission.biedi
The content of this file defines all the mission's objects, including units, triggers, waypoints etc. It is used when generating *mission.sqf* but also by the *VBS2 Mission Editor* when loading a mission in it. [22]

The files *mission.sqm* and *mission.biedi* are structured in the same way using the following format:

```
class Classname {
          variablename="value";
          class SubClass {
                    arrayVariable[]={"value1","value2"};
                    variableNumber=66
          }
}
```

In the extended *General Scenario Editor*, a method is written to simplify the construction of the above structure. The *mission.sqf* looks a bit different.

5.4.1.1 Waypoints

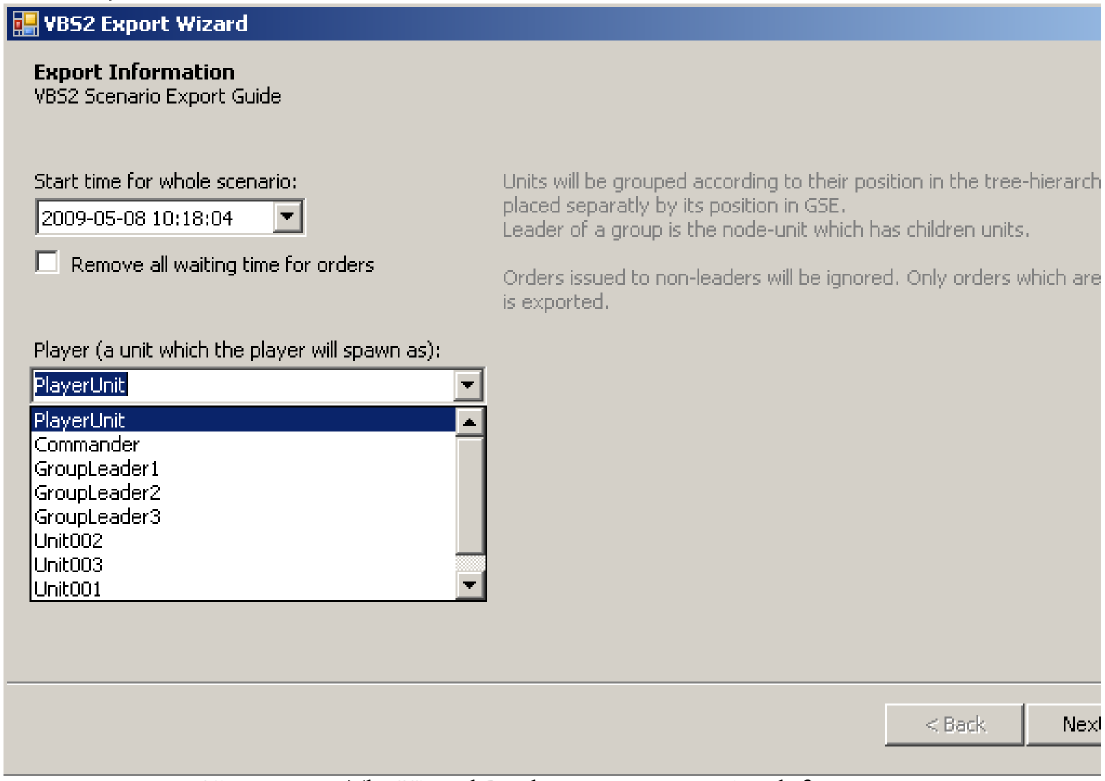
The tasks to be executed by the units of the scenario are represented by so called *waypoints* in the game. Units can be assigned to these waypoints, which in their turn can be connected to other waypoints. Specific settings can be applied to each waypoint.

All orders in *General Scenario Editor* are translated to waypoints in the game. Although the information about each order within the scenario editor is sufficient to build waypoints in the game, it is required to decide in which order these orders should be executed. This is done by sorting the orders by the time of their start condition.

5.4.1.2 Coordinates

The coordinates of the map is represented with simple (x,y,z) coordinates, where the altitude (z) often can be excluded to automatically place an object on the ground.
The (0,0) coordinate is positioned differently on every map, which makes the job of using a game-map in *General Scenario Editor* a bit complicated. No general method of determining where the (0,0) position is on the map could be found. The process of importing a map from the game into *General Scenario Editor* must therefore be done manually. The first step is to obtain an image of the map (tip: screenshot) and then identifying the coordinates of the image's corners. These corner-positions must then be entered when using the image in the scenario editor.

5.4.2 Export wizard


VBS2 Export Wizard

Export Information
VBS2 Scenario Export Guide

Start time for whole scenario:
2009-05-08 10:18:04

☐ Remove all waiting time for orders

Player (a unit which the player will spawn as):
PlayerUnit

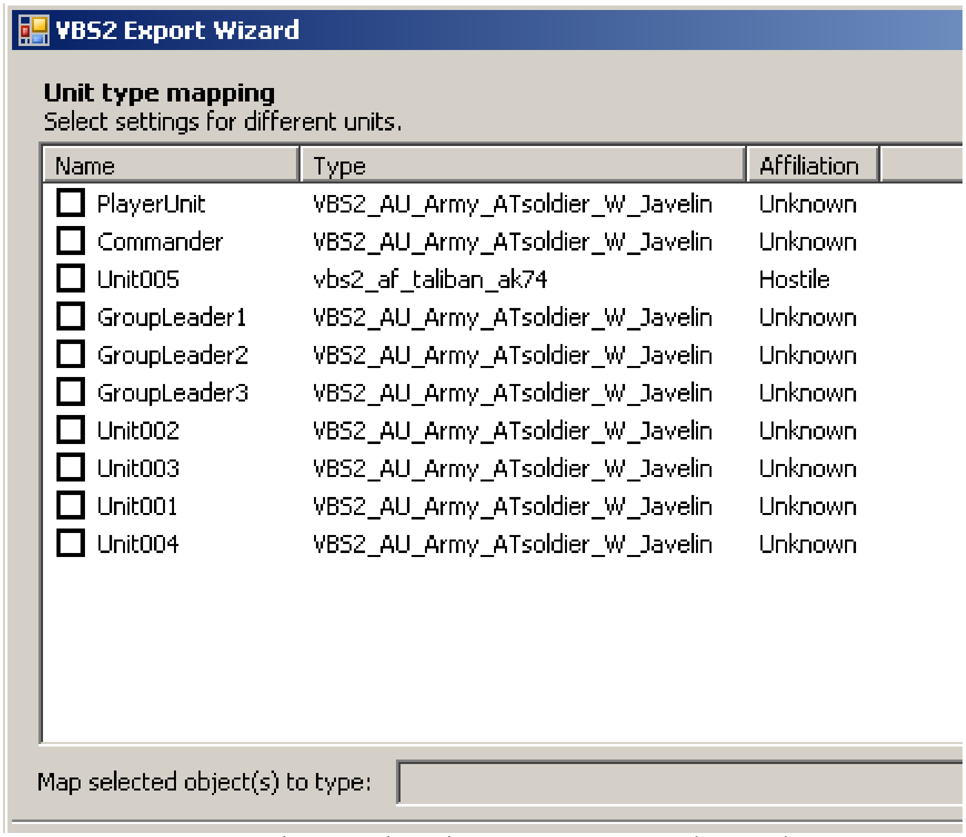
PlayerUnit
Commander
GroupLeader1
GroupLeader2
GroupLeader3
Unit002
Unit003
Unit001

Units will be grouped according to their position in the tree-hierarch
placed separatly by its position in GSE.
Leader of a group is the node-unit which has children units.

Orders issued to non-leaders will be ignored. Only orders which are
is exported.

Figure 5-11: The Virtual Battlespace 2 export wizard, first step.

The first step of the export wizard allows the user to define the start-time of the simulation. This time will be used as a reference when exporting the orders, which may have their own start-times set. As a consequence, this might induce some waiting time for some tasks. If this is not wanted, all such waiting times can be removed by checking the box below.
All scenarios in Virtual Battlespace 2 must include a playable unit. This unit is set using the combo-box.


VBS2 Export Wizard

Unit type mapping
Select settings for different units.

Name	Type	Affiliation
☐ PlayerUnit	VBS2_AU_Army_ATsoldier_W_Javelin	Unknown
☐ Commander	VBS2_AU_Army_ATsoldier_W_Javelin	Unknown
☐ Unit005	vbs2_af_taliban_ak74	Hostile
☐ GroupLeader1	VBS2_AU_Army_ATsoldier_W_Javelin	Unknown
☐ GroupLeader2	VBS2_AU_Army_ATsoldier_W_Javelin	Unknown
☐ GroupLeader3	VBS2_AU_Army_ATsoldier_W_Javelin	Unknown
☐ Unit002	VBS2_AU_Army_ATsoldier_W_Javelin	Unknown
☐ Unit003	VBS2_AU_Army_ATsoldier_W_Javelin	Unknown
☐ Unit001	VBS2_AU_Army_ATsoldier_W_Javelin	Unknown
☐ Unit004	VBS2_AU_Army_ATsoldier_W_Javelin	Unknown

Map selected object(s) to type:

Figure 5-12: The Virtual Battlespace 2 export wizard, second step.

The export wizard automatically assigns a "type" to each unit. This type is used by *Virtual Battlespace 2* to determine its model and is also dependent on the affiliation of that unit. In the future, it should be possible to assign different types manually in the export wizard. Such wizard step is prepared for use, as seen above.

5.4.3 Example of exportation to VBS2

Virtual Battlespace 2 is delivered with its own specific "mission editor". To be able to analyze the correctness of the exportation, a scenario was created in *General Scenario Editor* and exported to the game's mission files. The output material was then loaded into *Virtual Battlespace 2* own mission editor. An example of this procedure can be observed on the next page.

The first picture is taken from the scenario in *General Scenario Editor* while the second picture shows the result in the game's own mission editor. Although numerous differences between the pictures can be observed, the mission is actually being performed as intended.

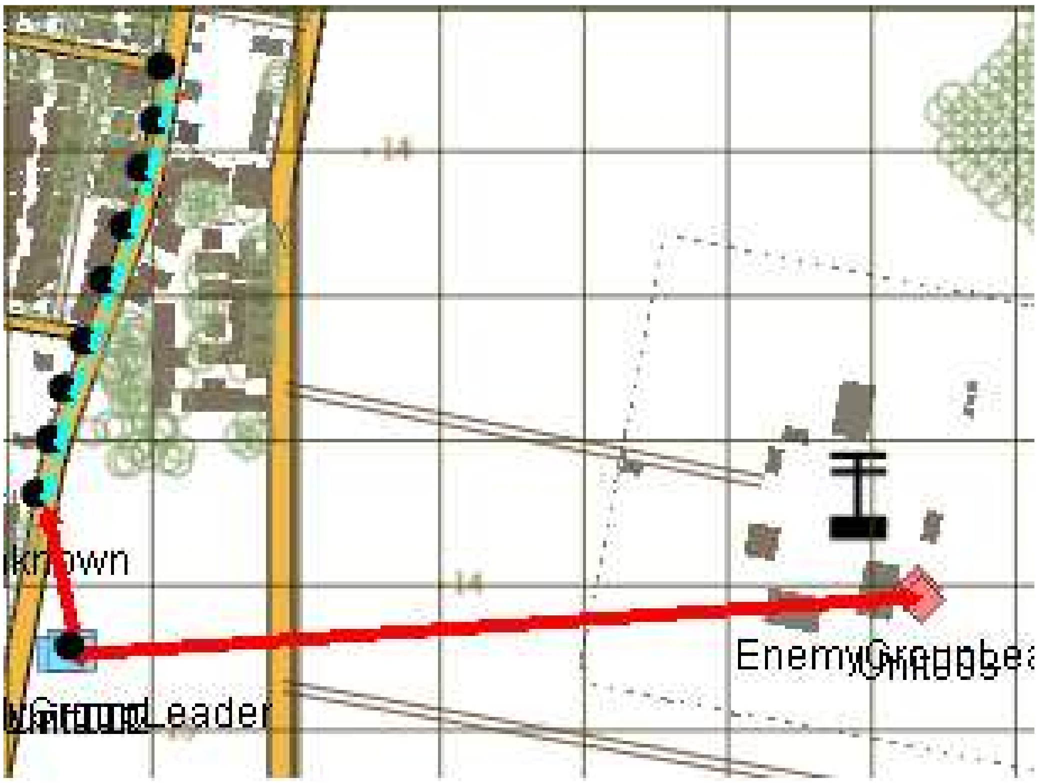


Figure 5-13: A screenshot of a scenario created in General Scenario Editor.

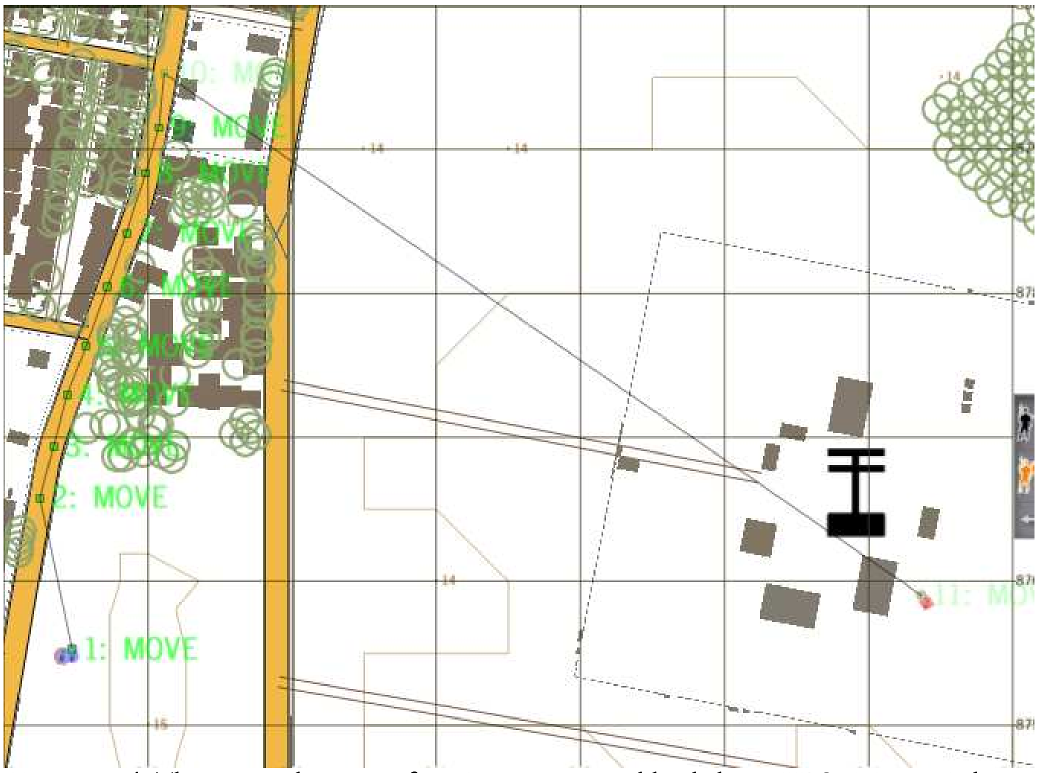


Figure 5-14: The exported scenario from Figure 5-13 and loaded into VBS2 Mission Editor.

5.5 Future extensions

During the development, many promising ideas have emerged. The following ideas have not been implemented but could be considered by future developers.

5.5.1 Future extensions to GSE

The following ideas are for *General Scenario Editor.*

5.5.1.1 Additional order types

There are many different types of orders that would be needed to make a versatile product. Looking at for example *Joint Consultation, Command and Control Information Exchange Data Model* (see chapter 1.3.3), there are very many action types possible in a military scenario. Since orders in *Battle Management Language* are constructed with a structured syntax, some of these orders are almost identical in terms of input from the user in the order editor. This would make it possible to add a considerable amount of new order types without making extensive modifications to the application.
For a detailed description of how new order types can be added, see appendix chapter 9.6.

5.5.1.2 Order modes

The scenario editor should be a *general* scenario editor. Thereby, it should be able to export several different formats, maybe different implementations of *Battle Management Language*.
Even if the input into the scenario editor should remain "general", some output formats would require that some specific input is done. To simplify the exportation of different output formats, the application could have different "modes" for orders.

An "order mode" functionality would define what information is required and what is optional for that specific mode. There could for example be a "JBML"-mode.

The contrary of modes, which is the current solution, is to specify all additional information in export wizards. This works well during small scale scenarios, but can be very time consuming in large scale.

5.5.1.3 View modes

During development in this thesis, it was assumed that the user views the scenario from a planner's perspective. This causes the operations and viewpoint to be "issuer"-orientated. It could however be of interest to some users to view the scenario from a performer's (*taskee*) perspective. This would affect the method of issuing orders but also the presentation of the timeline.
To be able to select between these perspectives, a "View mode" option could be introduced to the scenario editor.

5.5.1.4 Phases

The implemented timeline gives a time-influenced view of the orders within the scenario. The start condition of an order decides where on the time line it will appear. The start condition is either time or action based.

The commander of an operation might want to plan a mission according to different phases, with a specific intent for every phase. This approach could be used in the scenario editor as well. The user could for example define new phases and assign a set of orders to a specific phase. Some filter would allow the user to traverse between the different phases, and the order view would only display the orders within the selected phase.

5.5.1.5 Areas of interest

Similar to routes, there could also be a need for defining a specific area on the map. Areas would likely have the same type of input method as routes, described in chapter 5.3.1. The existing functionality to handle routes in the scenario editor can easily be adapted to also support areas. This is because the only difference between an area and a route is the way the coordinates are interpreted. Even the table for routes in the database supports areas without any modifications.

Regardless of routes and areas, some locations could be of special interest - drop zones, meeting point etc. The user might want to mark such position on the map and define it with a certain amount of data. Such positions could be called "areas of interest". These points could then be assigned as an order's target location.

5.5.1.6 Advanced tactical graphics

To visualize orders in a more overviewed way on the map, the scenario editor could support the appropriate use of the symbol set *2525B Appendix B*.
If all orders are specified correctly, all the data that has been collected can be used for choosing the appropriate tactical image, and to place it accordingly on the map.

5.5.1.7 Selective exportation using labels

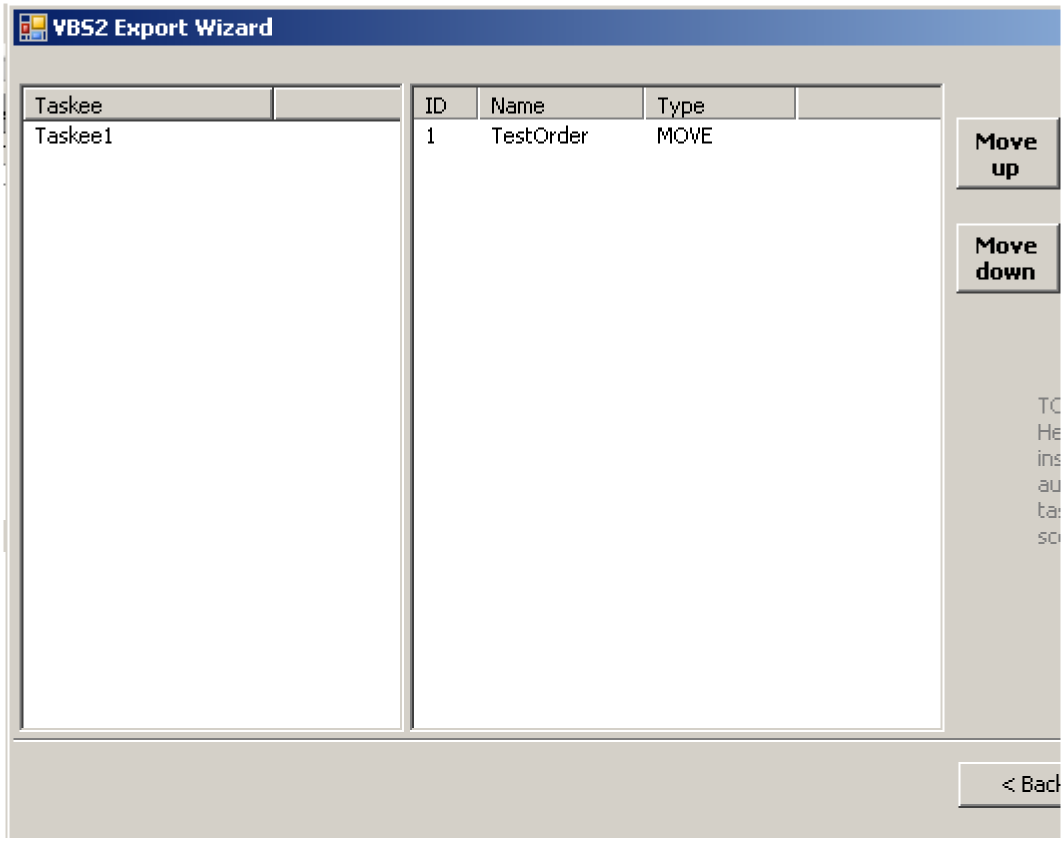
At the time of exportation, the user might want to specify different parts of the scenario to export. For example when exporting to *Virtual Battlespace 2*, only a subset of all the units in the scenario should be included in the output. This selection could be done completely in an export wizard but it would greatly simplify the matter if the user could define a set of labels that can be put on the different parts of the organization. During exportation, only entities with desired labels are included in the exported material.

5.5.2 Future extensions to VBS2 exportation

The following ideas are regarding the exportation to *Virtual Battlespace 2*.

5.5.2.1 Order of orders

A problem encountered during the implementation of the exportation mechanism was in determining the order in which tasks are executed. In *Virtual Battlespace 2,* all orders of a performer (taskee) are planned in a sequence. In *General Scenario Editor* however, orders are initially created independently of each other and can thereafter be assigned a "start condition". This means that the planner might have difficulties to determine in which order the orders will be executed when planning in *General Scenario Editor*.
This could be solved by letting the user explicitly define the order of orders, either in the export wizard or in another window of the application. An example how such interface would look can be seen in Figure 5-15.



Taskee
Taskee1

ID	Name	Type
1	TestOrder	MOVE

Figure 5-15: Example window of how the user could order the orders.

5.5.2.2 Identification of vehicles using symbols

It would be possible to automatically identify the model (type) to be used for each unit in *Virtual Battlespace 2*. This could be done for each unit by mapping its symbol from *2525B Appendix A* to a corresponding model in *Virtual Battlespace 2*.

6 Conclusions

6.1 Lessons learned

In the introductory research phase of this project, it was hard to identify any substantial differences between different *Battle Management Language* (BML) solutions available. At least any that directly would affect the work on the scenario editor. The goal was to find a common base of different languages and use it for designing the order structure in the scenario editor. It did not simplify the matter that every *Battle Management Language* seemed to aim for the same thing, or that some of the languages are just standards, while others are actual implementations.
After hours spent by reading about *BML* and all of its different variations, it became clear that the most logical approach would be to follow the 5 W's principle, which is a very convenient way for structuring orders. This principle is used by many of the encountered *BML*:s.
At the end it was decided to use *Joint Battle Management Language* (JBML) as a reference during development of the scenario editor. This decision was made in conjunction with the advisors recommendations but also backed up by the fact that JBML already had a design for implementation.

The 5 W's principle is used by the order structure in the scenario editor, simply because it is widely accepted and also proposed by the advisor. As the 5 W's is fundamental for all military tasks, the principle could be used as the base for all orders. The result is the *BMLActionType* class design in the application (see chapter 5.2.2.2). It is also very easy to map the 5 W's to positions on the map, in order to implement a simple way to draw the tasks. Basically, it is all a matter of identifying the "Who" and "Where" for the task, where the first specifies the start of the order arrow and the second specifies the end point.

Because *Military Scenario Definition Language* is a well formed standard for defining an organization it seems very natural to combine it with *BML* to initialize a scenario. As stated in the paper "An Application Extension for the Military Scenario Description Language", the connection between the two languages is very clear. *Coalition Battle Management Language* is being developed with *Military Scenario Definition Language* in mind, while even *JBML* uses its own modified version of the language. [8, 9, 17]
This connection is even more obvious when working with *Battle Management Language* in a military scenario editor. The organization management in the application was already implemented in the original *General Scenario Editor*. When the order management in the application was designed, the bridge between

the organization and its orders became very clear and many parts linked together perfectly.

The extensions made to the scenario editor were supposed to result in the possibility to export the scenario to *JBML*. To guarantee the success of this task, *XML* schema definition files for the language were used as a reference during design of the internal order structure. This ensured that the scenario would be convertible to *JBML*. And because *JBML* uses the 5 W's principle for orders, it is possible to export the orders to other languages as well.
Having the military doctrine behind *JBML* as a reference has influenced many decisions for the scenario editor. This introduced ideas that else would not had been thought of. For example without the guidance of *JBML*, the idea of implementing a way for specifying an indirect unit as a location for the order would probably not have been discovered. Some options available for an order, like the list of possible qualifiers for a start condition, were directly mapped from *JBML* to the order editor. The advantageous aspect of this is that the exportation can be done without any complex transformation between scenario editor and the *JBML* output. However, this has maybe drawn the development away from a "general" scenario editor, to a more specific *JBML* order editor at some points. This dilemma has introduced the idea of implementing "modes" as a feature in the scenario editor (see chapter 5.5.1.2).

6.1.1 Difficulties with JBML in a scenario editor

Academics at *George Mason University* are currently working on *IBML* which intends to correct some problems and limitations with *JBML*. [16, 21]
Some of these problems and limitations were encountered during the work on this thesis.

The perspective of the scenario editor changed during the design process, which affected the end result. The first vision was that orders should be assigned from an executer's view and not the issuer's. In *JBML*, a set of orders is always oriented towards a single issuer and transmitted in an "order push" (a set of orders). This means that all orders in an exported *JBML* output have to be from the same issuer. So with this in mind, the scenario editor is designed to require the issuer to be selected before the performing unit is specified. Another dilemma is that the scenario editor allows orders to be placed in relation to any other order in the scenario. If this relation spans over different issuers, this cannot properly be reflected in *JBML* output.

An order push in *JBML* is not only limited to a single issuing tasker, but also the issuing time of the whole order push. This property of *JBML* shaped the idea of introducing "phases" in the scenario editor as future extension. The idea is mentioned in previous chapter 5.5.1.4. Phases would make it possible to not only separate *JBML* output in the asp0ect of issuers, but also in a defined set of phases for the whole scenario. As a result, an issuer could have multiple order pushes with different issuing times for every phase.

A difficulty encountered with *JBML* was during the export according to the *XML* schema definition files. These files clearly stated what options that should be available for each attribute of an order, but did not specify what combinations of these properties that should be allowed. This makes it possible to create illogical and strange orders, where the properties contradict each other. So the question was whether to control the input to avoid such orders or to leave it at the user's responsibility to create proper orders. There is no documentation available that states which options that are reasonable combined with another. After consultation with the advisors, it was decided to leave the responsibility in the hands of the user.

6.2 The resulting solution

While some properties of an order are always needed for the task, there is some information that will only further enhance the detail richness of the order. Such information can be considered as "optional". Making the scenario editor as "general" as possible, the idea of offering optional properties in the order editor was proposed.

Optional properties can be used in a future implementation of "order modes". For example, imagine that the optional properties available represent parameters for a specific platform, such as a specific game. It is then still possible to create a general order, but also adding optional properties that only affect the exported material for a specific game. This would enable less information to be required in the export wizard during exportation.

Even if the order editor alone is enough to create and edit orders, it is still quite inconvenient to specify everything using a property grid (see terminology). To demonstrate a solution to this problem, some more user-friendly functions to edit an order is implemented using the map as interface. A right-click menu is added to the map, which is described in chapter 5.3.2.5.

The vision for visualizing orders on the map changed dramatically from the start of the project to the end result. At first, the proposed solution was to focus on tactical graphics for creating and visualizing orders, as described in chapter 4.1.2. More advanced components like *MOLE* and the symbol set *2525B Appendix B* were considered as means of representing and creating orders.
When it comes to the graphical representation of orders on the map, the end result is simpler than the first vision. Instead of advanced graphics on the map,

the focus was put on maintaining a clear overview of the scenario, and the simplicity of creating an order. Orders are represented by simple arrows on the map. This outcome is a consequence of the proposed solution described in 4.1.1. This choice was made after the initial examination of the components in *General Scenario Editor*. That examination showed that it would be very time consuming to modify the component that controls the map, to allow advanced graphics to be drawn on the map.

The first implementation for drawing orders on the map was not optimal when it came to show the scenario in a clear way. This was mainly because all orders of the scenario were simultaneously shown on the map, which gave an unreadable overview. A way of separating the view of the scenario in time would solve the problem. Also since the perspective of the order management is issuer (*tasker*) oriented; it also makes sense to only visualize the selected issuer's orders. The first proposed solution was to only implement a simple timeline. A timeline allows the user to select a time span, in which only the encapsulated orders are viewed. However it would be difficult for the user to understand what the timeline represents because it would not present the relation or extent of orders. This is the reason why a *Gantt* chart is added to serve both as scale for the timeline and to also give a more visualized overview of the orders regarding time. The result of this solution can be seen in chapter 5.3.3.

As stated in the problem description (chapter 2) there were two games of interest when it came to exporting a scenario from the application. These games, *Steel Beasts 2 Pro* and *Virtual Battlespace 2*, were both investigated. It was discovered that the mission files for *Steel Beasts 2 Pro* is in a binary and undocumented format, which makes exportation from the scenario editor virtually impossible considering the time frame of the last phase of the project. This led to the choice to follow up *Virtual Battlespace 2* as the target of exportation.

As none of the participants has enough experience regarding the principles of war fighting, it is difficult to make a realistic and advanced scenario translation to *Virtual Battlespace 2*. Even if the translation of military tasks was simplified, the result shows that the implementation of the order management was successful when exporting to the game. It is possible to create a more advanced translation if an exact definition of the different order types is used.

The interpretation of order assignments differs slightly between *General Scenario Editor* and *Virtual Battlespace 2*. In *General Scenario Editor*, the planner sets a start condition for each separate order, either dependent of time or another action. In *Virtual Battlespace 2* the orders are always issued one after another. Because the planner can combine action dependent and time dependent tasks in the *General Scenario Editor*, this makes the order of tasks uncertain. To solve this, a simplification was done, described in chapter 5.4.1. The way of managing orders in *General Scenario Editor* can achieve the exact same result as *Virtual Battlespace 2*. It however raises the question if *General Scenario Editor* should feature the possibility to organize the tasks in the order of execution. This is a decision that has to be made by future developers.

6.3 Project conclusions

At the beginning of this project, a great amount of information was revealed and processed rapidly. The most valuable information was found with the help of expert advisors, which made the research easier. Their aid during this project has been a crucial asset.

The project model and working base structure described in chapter 3 was followed quite well. As intended, weekly meetings were held with the advisor at *Saab Training Systems AB*. In this way, the development of *General Scenario Editor* was controlled as time went by. After the completion of the first and second task, the work was fully directed to third task as intended. After the completion of task three, there was some overall bug fixing of the entire solution for stabilizing *General Scenario Editor*.

As the original *General Scenario Editor* was written with C# as primary programming language, the extensions would also be written in C#. The fact that neither of the participants during this thesis had any previous experience with this language, led to some delays during development. This was however overcome after a small amount of time, when lots of similarities with Java were discovered.

Initially it was decided to use a “property grid” (see terminology) as the component for editing the properties of an order. Looking back, a more extensive inspection of *General Scenario Editor* and the property grid component should have been performed. Not saying that the choice of using a property grid is a bad decision, but many problems during initial implementation could have been avoided. These problems originated from the fact that the implementation was started before all the limitations of the property grids were known. For a more detailed description of the implementation of property grid, refer to chapter 9.3.

7 References

[1] Powers M., “A Geospatial Battle Management Language (geoBML) for Terrain Reasoning (I-110)”

[2] Kleiner M., “Standardizing Battle Management Language - A Vital Move Towards the Army Transformation”

[3] Tolk A., “Joint Battle Management Language (JBML) -
US Contribution to the C-BML PDG and NATO MSG-048 TA”

[4] Simulation Interoperability Standards Organization (SISO), “Standard for: Military Scenario Definition Language (MSDL)”

[5] Ullner F. and Lundgren A., “Lessons learned from implementing a MSDL Scenario Editor”

[6] Saabgroup, WISE Connectivy product description, (5 May 2009) <http://products.saabgroup.com/PDBWebNew/Generic.aspx?Entrance=Product&ProductCategoryId=274&ProductGroupId=394&ProductId=1593>

[7] Multilateral Interoperability Programme, “Overview of the C2 Information Exchange Data Model (C2IEDM) (C2IEDM Overview)”

[8] Blais C., “Coalition Battle Management Language (C-BML) Study Group Report”

[9] Tolk A., “An Application Extension for the Military Scenario Description Language”

[10] Multilateral Interoperability Programme official website about the organization, (5 May 2009)
<http://www.mip-site.org/011_Public_Home_Concept.htm>

[11] Hieb M., “Standardizing Battle Management Language – Facilitating Coalition Interoperability”

[12] Department of Defense, “Department of Defense Interface Standard, Common Warfighting Symbology”

[13] SCRUM official website, (5 May 2009) <http://www.controlchaos.com/>

[14] Jakob Blomberg at *Saab Training Systems AB*, there has been weekly meetings during this project.

[15] Per Gustavsson,

Date	Type	Description
2009-02-15 12:43	E-mail	First encounter with Per, got basic knowledge regarding *Battle Management Language* and *Military Scenario Definition Language.*
2009-02-16 14:42	E-mail	Received a great amount of documents that included information about: *C-BML Schema Definition files*, *Military Scenario Definition Language* and *Command & Control Lexical Grammar.*
2009-02-17	Meeting	During this meeting, several aspects, regarding which *Battle Management Language* that should be used, were clarified. Lots of information about the different BML:s.
2009-03-03 14:18	E-mail	Received a chart containing the Where-categories.
2009-03-05 15:20	E-mail	Clarification regarding *Command & Control Lexical Grammar* with different start-time/action qualifiers.
2009-03-23 09:15	E-mail	A discussion about difficulties concerning *Command & Control Lexical Grammar* and information about how to deal with problems in the Where attributes located in the 5 W's.

[16] Deepak Sumra,

Date	Type	Description
2009-03-19 20:18	E-mail	Got an answer about how the *Schema Definition Files* should be used regarding relations between orders and taskers.

[17] JBML official website, with *XSD* files downloadable, (5 May 2009) <http://netlab.gmu.edu/JBML/>

[18] Hieb M. et al., "Evaluating the Proposed Coalition Battle Management Language Standard as a Basis for Enhanced C2 to M&S Interoperability"

[19] US Department of Defense (DOD), “Geospatial Battle Management Language (GeoBML)”

[20] Hieb M., “Developing Extensible Battle Management Language to Enable Coalition Interoperability”

[21] Michael R. Hieb,

Date	Type	Description
2009-04-06 20:32	E-mail	Clarification about references between orders, also information about IBML was brought up in this e-mail.

[22] Bohemia Interactive Australia, “VBS2 Editor Manual 1.02”, (5 May 2009) <http://www.cs.adfa.edu.au/coursework/ZITE3107/documentation/VBS2_1_15/VBS2EditorManual.pdf>

[23] Official website of *ESRI*, developer of MOLE, (5 May 2009) <http://www.esri.com/software/arcgis/extensions/mole/index.html>

[24] An article concerning C2IEDM on the official website of The Forschungsgesellschaft für Angewandte Naturwissenschaften (FGAN – Research Establishment for Applied Science) (20 May 2009) <http://www.fgan.de/fkie/fkie_c40_f12_en.html>

[25] Salisbury M., “Command And Control Simulation Interface Language (CCSIL): Status Update”

[26] Official website of Virtual Battlespace 2, (5 June 2009) <http://virtualbattlespace.vbs2.com>

[27] Official website of Steel Beasts 2 Pro, (5 June 2009) <http://www.steelbeasts.com>

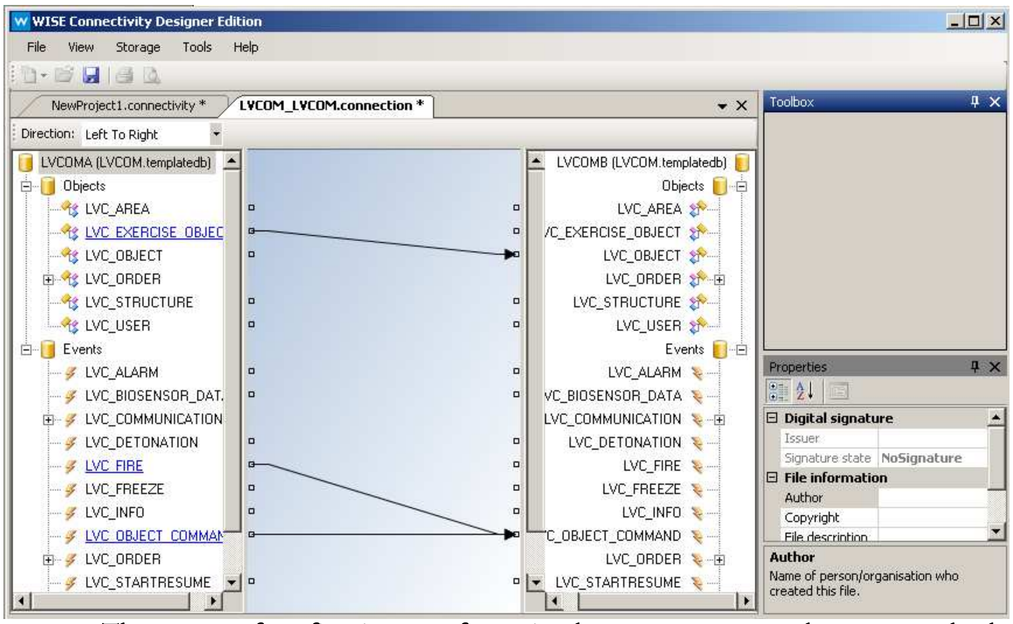
8 Terminology

Abbreviation/Expression	Meaning/Definition
2525B	See *MIL-STD-2525B*
5 W's principle	Where, When, What, Who, Why. A principle for structuring an order.
BML	Battle Management Language. Defined as: *"BML is 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."*
C2IEDM	A data model used for interoperability between military systems.
C-BML	*Coalition Battle Management Language*. The standardization of *BML*, currently under development.
Command & Control	Often abbreviated *C2*. *Command & Control* is a term to describe the routine of a commanding officer to assign tasks to forces in a military mission.
Command, Control, Communications, Computers, and Intelligence	Often abbreviated *C4I*. C4I can be summarized as C2, but adding the communication with computers and military intelligence.
Export wizard	A graphical guide for collecting required user input, which will then be used during an export procedure.
Gantt chart	A type of bar chart.
GSE	General Scenario Editor (from *Saab Training Systems AB*).
JBML	Joint Battle Management Language.
JC3IEDM	See C2IEDM.
MIL-STD-2525B	A set of symbols, used for military representation. Includes several appendixes, for different purposes. Appendix A – Units and resources. Appendix B – Tactical representations.

MOLE	Military Overlay Editor. Is a set of COM components for developers to create custom applications that support Department of Defense (DoD) MIL-STD 2525B and NATO's APP6a specifications.
MSDL	Military Scenario Definition Language.
Order editor, order editor window	The window in the extended GSE application which allows the user to create orders in the scenario.
Order push	A collection of orders (tasks) that is sent by an issuer.
Original General Scenario Editor	The version prior to the extensions made during this project.
Extended General Scenario Editor	The version after the extensions made in this thesis.
Property grid	A graphical component in Visual Studio.
Route	A defined set of waypoints used for traveling.
SB2	Steel Beasts 2 Pro.
Scenario editor	An application used for planning a military scenario.
Timeline	The graphical component in GSE that allows filtering of order by time.
VBS2	Virtual Battlespace 2
Who: Affected	The one that is affected by the order.
Who: Executer	The one that executes the order.
Who: Taskee	The one that receives the order.
Who: Tasker	The one that gives the order.
XSD	XML Schema Definition. Defines the structure of a XML file.

9 Appendix

9.1 WISE Connectivity



Direction: Left To Right

Digital signature

Issuer	
Signature state	NoSignature

File information

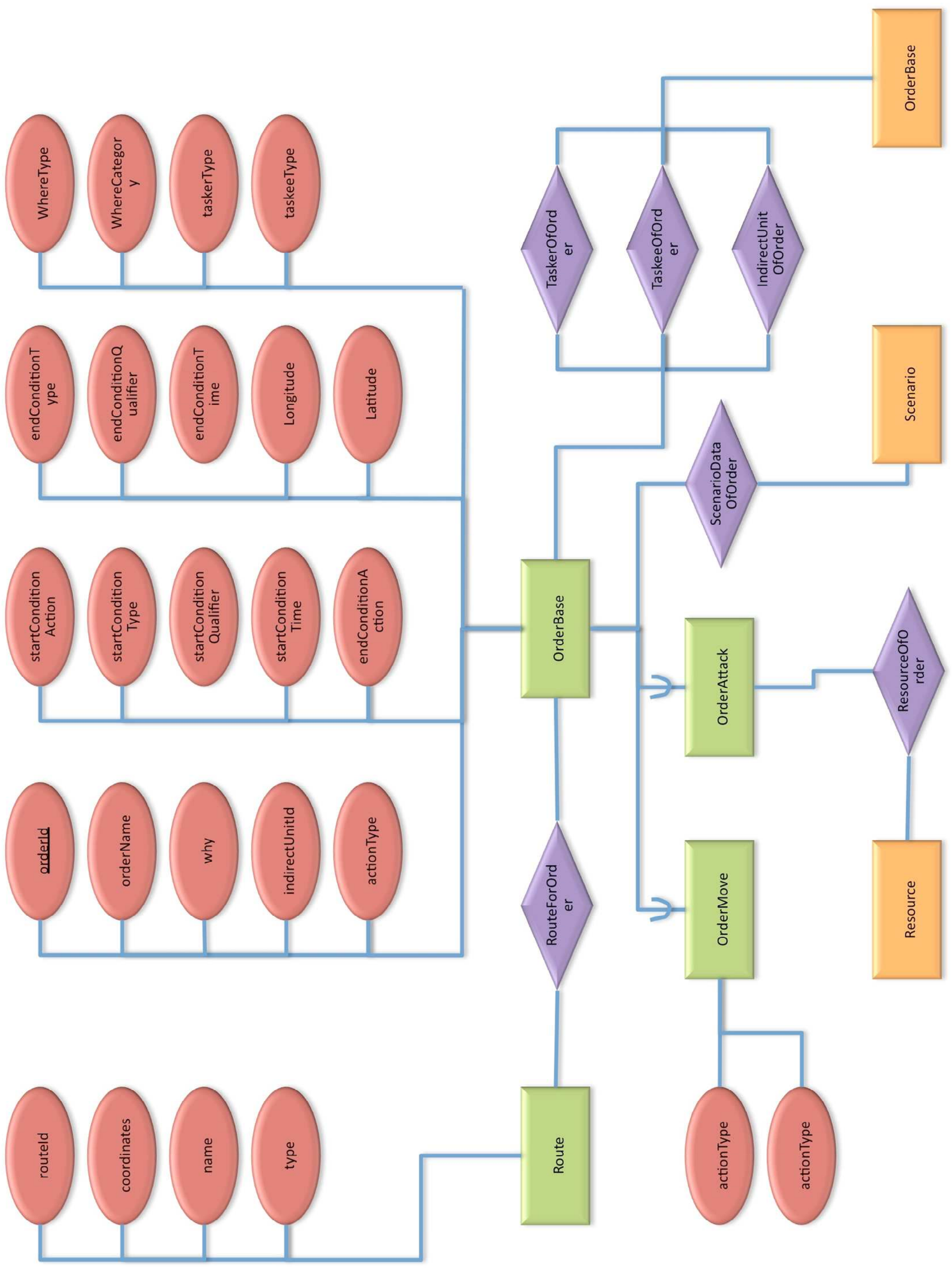
Author	
Copyright	
File description	

Author
Name of person/organisation who created this file.

Figure 9-1: The process of configuring transformation between a system and a common database.

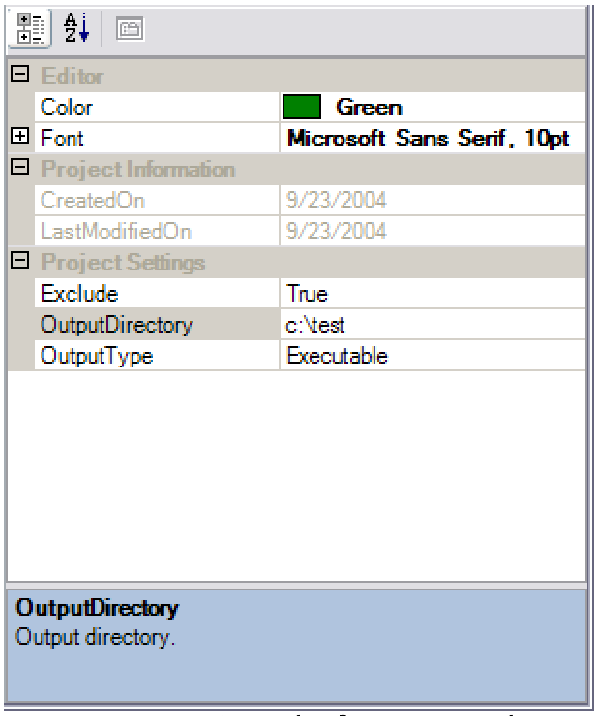
The traditional approach for linking several different systems is to modify the participating systems to make a common interface for communication. This method is costly and time-consuming which is why Saab Training Systems AB developed *WISE Connectivity*. The main idea is to enable the developer to "configure" the integration instead of using programming as the primary tool [6]. To achieve this, *WISE Connectivity* does not require modification of the current systems, but instead make the integrations into a common backbone system, leaving the systems intact. The only programming actually needed is the driver for the specific system, linking it to the backbone. The rest of the work mainly consists of configuring the setup and the information model, which enables data to be shared between the systems [6].
Figure 9-1 shows the part of the configuration where an information model for a specific application is linked and transformed to the common information model for all participating systems.

9.2 E/R-diagram



9.3 The property grid

Microsoft Visual Studio 2008 provides a graphical component called a “Property grid”. This can be used as a graphical user interface to specify values to an object. During examination and study of the original version of *General Scenario Editor*, the use of this component could be observed in several solutions in the application. It also seemed to be a very convenient component to specify settings for an order.



⊟ Editor	
Color	🟩 Green
⊞ Font	**Microsoft Sans Serif, 10pt**
⊟ Project Information	
CreatedOn	9/23/2004
LastModifiedOn	9/23/2004
⊟ Project Settings	
Exclude	True
OutputDirectory	c:\test
OutputType	Executable

OutputDirectory
Output directory.

Figure 9-2: A sample of a property grid

A “property” for an order is a defined attribute for some part of an order. For example the number 44,322 is assigned to the “Longitude” property.
The property grid component lacked a few dynamic aspects that would be needed for the order management in the extended *General Scenario Editor*. Therefore a certain workaround, or extension, is made.

9.3.1 Groups

The first requirement was to allow grouping of different properties for an order. For example a “Coordinate”-parameter for an order requires both a Longitude- and Latitude-property. As a result of this, a simple data structure, *WISEPropertySpecGroup*, is created which basically contains a list of *WISEPropertySpec* objects.

9.3.2 Special combo-box

In an order, there are some properties which rules out other ones, and vice versa. An example would be that if a coordinate is specified as the location of a target, there cannot be another type of property that specifies the location the target, as that would cause ambiguity. The solution is to create a special type of combo-box. When one option is selected in the box, several new properties appear in the property grid, and the old ones are removed. This was solved by creating a subclass of *WISEPropertySpec*, to maintain compatibility with other methods. The new subclass was named *WISEPropertySpecCombo*. Several *WISEPropertyGroup* objects can be added to an instance of this class. The different groups are inserted into the property grid depending on the selection of the combo-box.

```
//////////////////////////////////
//TASKEE
//////////////////////////////////
WISEPropertySpecCombo comboTaskee = new WISEPropertySpecCombo(
           this,
           "Taskee type",
           "Who",
           "Taskee type",
           "");
WISEPropertySpecGroup groupComboTaskeeType = new
WISEPropertySpecGroup("Taskee type");
groupComboTaskeeType.Add(comboTaskee);

WISEPropertySpec propTaskeeUnit = new WISEPropertySpec(
           this,
           "Unit (taskee)",
           typeof(UIListBoxEditor),
           "Who",
           "Unit",
           "",
           typeof(UIListBoxEditor),
           typeof(System.Drawing.Design.UITypeEditor));
WISEPropertySpecGroup groupTaskeeUnit = new
WISEPropertySpecGroup("Unit");
groupTaskeeUnit.Add(propTaskeeUnit);

comboTaskee.Add(groupTaskeeUnit);
_specGroups.Add(comboTaskee.Name, groupComboTaskeeType);
groupComboTaskeeType.Mandatory = true;
```

Figure 9-3: Code example of both *WISEPropertySpecCombo* and *WISEPropertySpecGroup*.

9.4 The order editor, technical details

9.4.1 How the order editor handles orders

All created orders require an instance of a *BMLActionType*-object, or a class derived from it. The most information about the task is contained within this class. A new instance of the class is created when the user chooses the type of order in the order editor window. It is actually a *BMLActionType* object that is edited in the order editor's property grid. The order editor has a special procedure when loading a new action type, as follows.

When a *BMLActionType* is loaded in the order editor, the first thing needed to be done is to setup its property specifications. This only needs to be done once with the method *updateProperties()*. The method simply defines all the property specifications groups (*WISEPropertySpecGroup*) and their containing specifications (*WISEPropertySpec*) for the action type.

When the specifications are defined, the action type is processed by the order editor and adds the property groups and their property specifications to the property grid, visible to the user. Only the property groups that have their inserted-flag raised will be added to the property grid. At this point, all properties in the grid have a *null* value, or maybe the default value of its type.

Afterwards, the member method *initializeProperties()* is called for the action type. This method assumes that all needed property specifications have been added to the property grid, and assigns values to these properties, according to the values from the member attributes of the *BMLActionType* object. This makes it possible to set default values to the property grid when a new order is created. But it also enables synchronization of saved variables from an *BMLActionType* fetched from the database.

When the user changes a property in the property grid, this update must also be mapped to the objects member attributes. This is done using the *UpdateProperty()* member method. The order editor calls this method when property has been changed. As this is event triggered, the property grid and the member attributes of the *BMLActionType* are always synchronized.

When the user wants to save the order, either by creating a new one or saving changes for an existing order, the order must be packaged in a *BMLOrder* instance. Checks are made so that no mandatory property has been left unfilled by the user. The method *createOrder()* returns a packaged *BMLOrder* instance containing all the options specified in the order editor, ready to be transferred to the database.

9.4.2 Dynamic lists in the order editor

For some of the properties in the property grid for an order, their values might only be a certain set of options. For example, if a “Target”-property should list all the units of the scenario, this list must be updated when a unit is added or removed from the scenario. Therefore a method, *maintainLists()*, in the order editor was written to automatically update affected lists present in the property grid. The method is also used as an initialization procedure for properties needing a dynamic list.

9.5 The BML export wizard, technical details

The wizard is started when an instance of the *Military Scenario Definition Language* export handler (*MSDLHandler*) is created. When the wizard is finished, all information needed for the export has been gathered and the exportation can begin. One of the following cases will be executed depending on the choice of export mode that has been made in the first step of the wizard.

- If the output should be *Military Scenario Definition Language* only, just call the original Export-method in the *Military Scenario Definition Language* export handler.
- If the output should be *JBML* only. Create an instance of *BML* export handler (*BMLHandler*) and call its Export-method.
  The Export-method will be aware of the options and will not include any organization information in its output.
- If the output should be *JBML* and *Military Scenario Definition Language 1.0* in the same file, an instance of the *BML* export handler will be created and its Export-method will be called. The Export-method will be aware of the options, and call *getOrganizationPart()* from *Military Scenario Definition Language* export handler to generate appropriate *MSDL* output.
- If the output should be *JBML* and *Military Scenario Definition Language 1.0* in separate files, call the original Export-method of both export handlers. The *BML* export handler will be aware of the options and will not include any organization information in its output.

9.6 For further extensions

This chapter should not be seen as a detailed user guide for further extensions. It merely provides a "checklist" for things to do when extending the order management in *General Scenario Editor* based on the current implementation.

9.6.1 Adding a new order type to the internal structure

As described in chapter 5.2.2, the implementation uses the *BMLActionType* as the base for the representing all tasks. Two inheriting classes, *BMLMoveActionType* and *BMLAttackActionType* are added to demonstrate the functionality of the internal order structure in the extended *General Scenario Editor*. Different order types are then assigned directly to *BMLActionType* or specifically to one of its subclasses. The only difference between orders belonging to the same class is the name of the order type, such as "ATTACK" or "STRIKE".

If a new order type is needed, and its properties equals to one already existing *BMLActionType*-class, it is only a matter of defining the new order type name and assigning it to an existing class in the order editor source code. However, if the new order type requires new types of settings, one will need to further inherit *BMLActionType* or one of its subclasses and add the new properties to it.

For example, a new order type is needed. The new order type is almost identical to the existing "ATTACK" order type, but requires one extra property called "Rate of fire". The developer can then easily create a subclass which inherits *BMLAttackActionType* in which he or she defines the new property "Rate of fire". All other properties would be inherited from the base classes. A new property also requires a new field in the database for storage. As modifications to the database are made, also the interface for fetching and saving orders from the database (*ScenarioDataAccess*) will need to be extended as well.

9.6.2 Adding new properties to order types

When adding or changing a property to a *BMLActionType*-class, there are several places in the source code to consider.

- The *updateProperties()* method, which defines the properties.
- The *initializeProperties()* method, which sees to that the properties visible in the property grid is gets the actual values of the *BMLActionType*'s attributes.
- The *UpdateProperty()* method, which sees to that all changes in the property grid are synchronized to the *BMLActionType's* attributes.
- All methods regarding order fetching, creation and updating in the database interface (*ScenarioDataAccess*) needs to be updated to support the new database table fields for the new property.
- If the property is a new type of dynamic list (9.4.2), the method *maintainLists()* may need to be modified to support it.