Lessons learned from implementing a MSDL Scenario Editor

A tool for the serious gaming community

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg
Computer Engineering

Bachelor thesis:
Fredrik Ullner
Adam Lundgren

© Copyright Fredrik Ullner, Adam Lundgren

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 2008

Abstract

Serious gaming is the collective name for the utilization of games and tools for education purposes. The utilization allows the organizations to train more effectively in various simulations and simulators. The simulations can be conducted in a high detail rich environment, without having to physically perform the exercise or simulation, for a lower cost.

This thesis describes the lessons learned when implementing the version one Military Scenario Definition Language (MSDL) standard (SISO, 2008) in a novel and proof-of-concept tool for serious gaming scenario interaction. The tool is able to create, import, export and validate military scenarios in MSDL form, as well as exporting scenarios in Open Geospatial Consortium Keyhole Markup Language (OGC-KML, 2008) form and in Virtual Battlespace 2 script files, importing a map through vector files, support for MIL-STD-2525B (Department of Defense, 2005) and much more. The tool is divided in three parts; graphical user interface, back end and scenario module, where each part is separated from the others with own tasks.

Comments of the current MSDL specification and suggestions for future revisions are included; while the current specification is easy to read and follow, quirks are identified and proposed solutions are presented. Examples of scenarios in MSDL form are included in the appendix.

This thesis also describes a common reference model for serious gaming and a common reference model from a real-time strategy (RTS) game perspective. The common reference model for serious gaming describes the most common information possible to gather in serious games. The common reference model serves as a base for applications and games for their information exchange; any information exchange shall have a corresponding entity or attribute in the common reference model. The RTS common reference model is based on a commander's perspective and the information such commander is exposed to, and is likewise a base for RTS games' information exchange.

This thesis is a joint assignment by Lund University, LTH and Saab Training Systems. The common reference model for serious gaming, the proof-of-concept application and exporting to MSDL were the acceptance requirements of this thesis, for Saab Training Systems. All acceptance requirements are completed as well as other additional tasks, such as; importing MSDL scenarios and exporting as OGC-KML and Virtual Battlespace 2 script files.

Keywords: serious gaming, military scenario definition language, MSDL, editor

Sammanfattning

Seriösa spel och syften är det sammanbindande namnet för användningen av spel och verktyg för utbildningssyften. Användningen tillåter att organisationer kan träna effektivare i olika simulationer och simulatorer. Simulationerna kan utföras i en miljö med högdetaljerad omgivning, utan att behöva fysiskt utföra övningen eller simulationen, för en lägre kostnad.

Detta examensarbete beskriver de lärda kunskaperna från att ha implementerat version ett av Military Scenario Definition Language (MSDL) standarden (SISO, 2008) i ett verktyg för interagering med seriösa spels scenarier. Verktyget kan skapa, importera, exportera och validera militära scenarier baserade på MSDL-form, samt att exportera scenerier på Open Geospatial Consortium Keyhole Markup Language (OGC-KML, 2008) och Virtual Battlspace 2 script-fil, importera en karta via vektorfiler, support för MIL-STD-22525B standarden (Department Of Defense, 2005) och mycket mer. Verktyget är indelat i tre delar; gränssnitt, motor och scenariomodul, där varje del är separerad från de andra och har egna uppgifter.

Kommenterar om den nuvarande MSDL-specifikationen och förslag för framtida versioner är inkluderat i examensarbetet. Även om den nuvarande specifikationen är lättläst och lätt att följa, har problem identifierats och förslag på förbättringar är presenterade.
Exempel på scenarier i MSDL-form är inkluderat i appendixet.

Detta examensarbete beskriver också en generell informationsmodell för seriösa spel och en informationsmodell för realtidsstrategispel. Den generella informationsmodellen beskriver den mest förekommande information i seriösa spel. Modellen agerar som en bas för applikationer och spel för deras informationsutbyte; all informationsutbyte skall ha en korresponderande entitet eller attribut i den generella informationsmodellen. Realtidsstrategiinformationsmodellen är baserad på en befälhavares perspektiv och den information befälhavaren är utsatt för och är liknande en bas för realtidsspels informationsutbyte.

Detta examensarbete är ett samarbete mellan Lunds Universitet, LTH och Saab Training Systems. Den generella informationsmodellen, MSDL-verktyget och export av MSDL var acceptanskraven för detta examensarbete, ställda av Saab Training Systems. Alla acceptanskraven är uppnådda, samt importering av MSDL scenarier och exportering av OGC-KML.

Nyckelord: seriösa spel, military, scenario definition language MSDL, redigerare

Foreword

This paper and delivered software comprises the bachelor thesis about Lessons learned from implementing a MSDL Scenario Editor – A tool for the serious gaming community. The thesis is a joint assignment by Lund University, LTH and Saab Training Systems. The work was performed at Saab Training Systems' facilities in Helsingborg, Sweden.

To our assistance, we have had Christin Lindholm as the thesis examiner, and Jakob Blomberg and Niklas Andersson as the technical supervisors.

We would like to thank Per Gustavsson at Saab Group for his input regarding MSDL and the employees at Saab Training Systems in Helsingborg. We would also like to thank John Olsson and Robert Michalski for their cooperation.

List of contents

Introduction

The video game industry is one of the largest industries in the world as described in the article Video Games Group Up (Anon). Each game developed serve a purpose, allowing its users to interact with the game, learn from the game, interact with other humans through the game or some other purpose the authors intend. Researchers have found that children may benefit from educational aspects from games, especially now in our digital age (Anon, Computer Games – Education). In 1980, video game producer Atari released a game called Battlezone (Anon, Battlezone) which took education to a higher level, designed for adults and contained military perspectives. While the game itself wasn't designed for military (around the world), but as a simple arcade game, it paved the ground for the concept called serious gaming (Robert Stone, Education and Training – Serious Gaming, and Anon, 2008, Gaming Technologic Impacting Military Training).

1 Background

What exactly is serious gaming? It is a concept, entailing using computers, games and simulation exercises tweaked and primarily for military organizations. Basically, the serious gaming genre mean that a game or tool shall be used by military organizations, allowing people to train and have fun in various simulations or simulators, in a high detail rich environment, without having to physically perform the exercise, with lower cost.

As a military organization is governed by a set of people and the organization may contain many levels of management, all computer games or simulation services are not practical for every level or for every person. As such, tools and games are created for each individual level, allowing multiple people of different branches to interact. That is, a first-person shooter game may be suitable for a person training to be a tank driver, while a real-time strategy game may be more suitable for a commander (of many tanks), allowing each individual the level of abstraction needed to perform their duties.

As different games or tools may interact, knowing what and how to communicate is crucial. A game aimed at Swedish forces may need to train with a game that American forces are using, and as such the games (and tools) need to be able to communicate with a defined set of instructions both games understand. The two games need to use similar symbols to denote “tank” respectively “driver”, and so on.

The High Level Architecture (IEEE-1516, 2008) (from here on referenced as HLA) platform is a wide spread and flexible implementation of such communication, designed for military simulations to interact with other games. However, HLA’s flexibility may cause issues, as is noted in Petersson, High Level Architecture.

1.1 Project background

HLA’s flexibility issues have turned organizations to research after other technologies and specifications. The newly developed Military Scenario Definition Language (SISO, 2008) by Simulation Interoperability Standards Organization (from here on referenced as SISO) is such a specification. The intent of MSDL is to allow scenarios and simulations to be defined in a standardized manner, regardless of origin.

For a standard to be successful it needs a user base that can adopt the standard and perform experiments. MSDL scenario generation as well as validation of MSDL scenarios has become important. The currently known tool capable of

rendering, generating or using the MSDL standard is built upon Microsoft PowerPoint (OneSAF, OneSAF's MSDL implementation), and is not available to a wide user base.
Saab Training Systems (from here on referenced as STS) intend to allow that organizations and people use MSDL for scenario generation and rendering through an initial proof-of-concept application. STS also wish to develop a general and a real-time strategy game reference model for game interaction in military and civilian domains.

1.2 Problem description

Each game has its own information model and each game has its own communication protocol. When integrating games to other systems or other games the information models as well as the communication protocols needs to be aligned. STS has a framework that enables integration of various protocols and information models. However, for each new game that is to be added to the STS systems flora there is a need that the information model has to be made specifically for that game, see figure 1A.

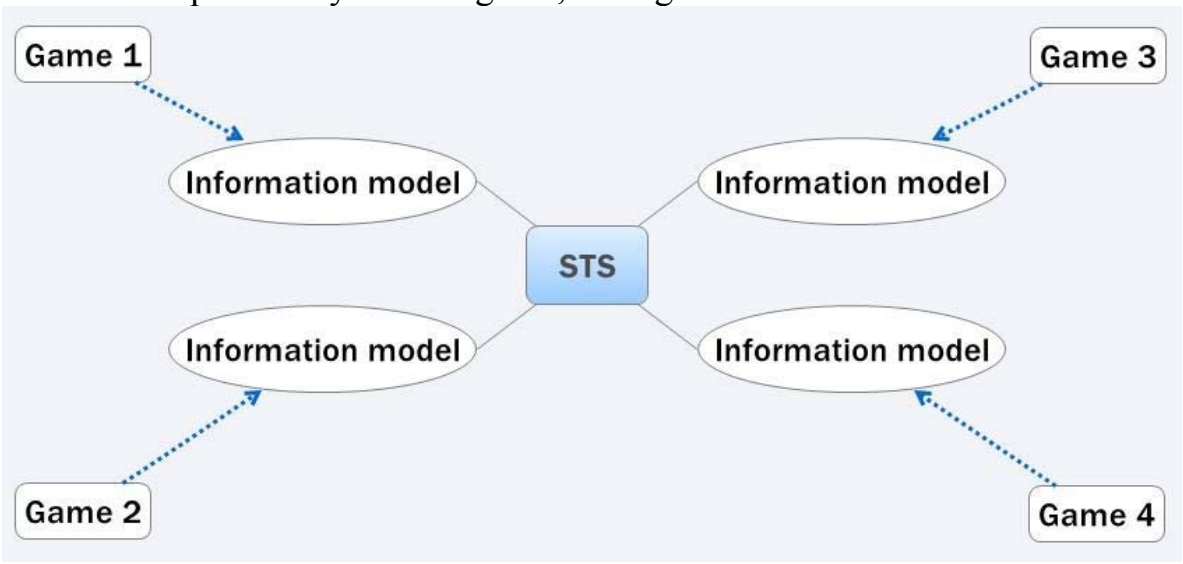

Figure 1A – The old reference model scheme

To reduce the information model creation for each game the intent is to develop a common reference model (CRM) that fits most of common games, as can be seen in figure 1B.

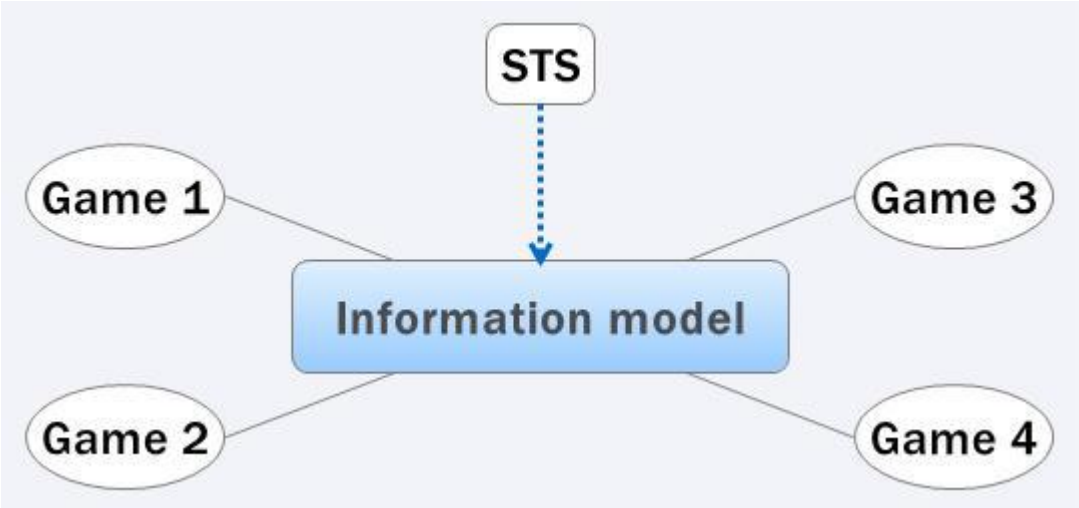

Figure 1B – The new common reference model scheme

The first objective in this thesis is to create a reference model from a tactical (real-time strategy or third-person) game point of view. That means that the starting point is to look from the operational perspective, i.e. the commander's view, and determine what information is most important for the commander and what information shall be sent and retrieved. The second objective is, together with John Olsson and Robert Michalski (Olsson, Michalski, 2008) (from here on referenced as "cooperation thesis"), present a serious gaming common reference model that fits both first-person shooter games and third-person tactical games.

The third objective is to develop a MSDL scenario editor. The editor must generate, import and export MSDL scenarios based on the MSDL specification. The parts of the MSDL specification that relates to the STS object model and the identified Common Reference Model for Serious Gaming will be implemented, thus leaving some parts of the MSDL specification un-implemented.

As the MSDL specification has been newly developed, problems and issues regarding the MSDL specification and its implementation will most likely arise. As such, the forth objective is to document the issues and problems and when found proposals for solutions.

1.3 Goals

1.3.1 Saab Training Systems goals

STS long term goal is to increase the relation between themselves and Lund University, LTH.
STS' goals with this project are;

- Expand the range of STS products to offer a more total training solution.

- Simplify the preparations for training sessions, by using the MSDL standard. The system should be able to generate, import and export training scenarios on MSDL format and then be able to load that scenario file in all games and systems connected.
- A Strategy Serious Gaming Common Reference Model (SSGCRM) shall be created for real-time strategy based games, such as World in Conflict or Command & Conquer.
- A Serious Gaming CRM (SGCRM) shall be created for all games, together with the cooperation thesis. The SGCRM shall describe the lowest common denominators for games in respective genre.

1.3.2 Acceptance requirements

STS' acceptance requirements for this project are;

- Create a common reference model for serious gaming and a common reference model for real-time strategy games.
- Create a proof-of-concept application that can generate scenarios in MSDL format.
- Export of scenarios in MSDL form.

Lund University and LTH's requirement are to create a thesis paper based on the experiences and knowledge in the project, as well as the completion of this project and course.

See chapter 7 – Results – for acceptance requirement resolutions.

1.3.3 Our project, thesis and course goals

The goal of this project, thesis and course is to use all previous knowledge and incorporate it into this thesis paper, minimally fulfilling the acceptance requirements in section 1.3.2, and completing the course.
Furthermore, our goals include;

- Additional functionality to the application in the form of satellite image rendering; standardized symbology; extended import and export functionality; a simple code base for further development.
- Creating a run-down on the information gathered and lessons learned while implementing the MSDL specification.
- Allow the committee in charge of MSDL to see the specification from a different point of view, leading to improvement in MSDL.
- Become co-writers in a paper for the MSDL standard, which will be presented at a SISO Simulation Interoperability Workshop (SIW) in August of 2008.

2 Work methods

This chapter will describe the project algorithm for the different project phases, the project model chosen and why we chose it, time structuring of the project and a description of the tools we used.

2.1 Algorithm

This project had four work phases, as can be seen in Table 2A.

Phases	
	SUB PHASES
1. Information gathering	Reference modeling Elicitation of MSDL
2. Production	Design Implementation Testing
3. Presentation	
4. Thesis	

Table 2A – Work phases

2.1.1 Phase 1

This phase, *information gathering,* can be seen in Table 2B. In this phase, the SSGCRM and SGCRM will be created and elicitation of the MSDL specification for valuable information.

Information gathering	
	DESCRIPTION
Reference models	Create two reference models, one for real-time strategy (RTS) based games and one common reference model, for all games. The reference models shall describe entities from respective perspective, the entities attributes and entity relationships. The common reference model shall be created together with cooperation thesis (Olsson, Michalski, 2008). See section 3.1 and 3.2 for the RTS reference model and relationship model and section 3.3 and 3.4 for the common reference model and relationship model.
Elicitation of MSDL	As we may not have time to implement the entire MSDL specification, we shall elicit those parts that are of interest to STS and the MSDL community. As well, this time shall be used to gather information about MSDL and gain knowledge about the specification, before any attempt at implementing is initiated. See chapter 4 for MSDL information and commentary.

Table 2B – Information gathering

2.1.2 Phase 2

This phase, *production*, contains three sub-phases, as can be seen in Table 2C. In this phase, the scenario editor will be designed, implemented and tested. The sub-phases are iterative in their nature, allowing us to move between the sub-phases seamlessly.

Production	
	DESCRIPTION
Design	This production phase shall be used to design the MSDL Scenario Editor. This phase will focus on design for the database, interface, general handling of MSDL and all components interaction and communication. Only when the design for a feature is completed shall the project move to the implementation phase. See chapter 5 for the design details.
Implementation	This production phase shall be used to implement the design decisions done in the previous phase. If there are issues with implementing the specified design, the project shall move to the drawing board and redesign said feature. Only when the implementation for a feature is completed shall the project move to the testing phase. See chapter 6 for implementation details.
Testing	This production phase shall be used to test the implemented features completed in the previous phase. If there are issues with testing or if testing fail, the project shall move back to the implementation phase and possibly design phase.

Table 2C – Production

2.1.3 Phase 3

This phase, *presentation* can be seen in Table 2D. In this phase, the thesis' presentation will be prepared and performed. The presentation for Lund University, LTH, will be conducted at Lund University, LTH, with focus on the work process of the project. The presentation for STS will focus more on the technical details of the project.

Presentation	
	DESCRIPTION
Presentation	**Create and hold two presentations; one for STS and one for Lund University, presenting our thesis.**

Table 2D – Presentation

2.1.4 Phase 4

This phase, *thesis* can be seen in Table 2E and is parallel with the other phases. In this phase, this thesis document will be compiled.

Thesis	
	DESCRIPTION
Thesis	This thesis, containing gathered experiences and information, shall be delivered to STS and Lund University.

Table 2E – Thesis

2.2 Project model

Our main task was to develop a graphical scenario editor. Neither of us was familiar with creating graphical interfaces and therefore an agile (Anon, Benefits of Agile Development) model – extreme programming – was a better choice rather then e.g. the waterfall model (Anon, 2007, What is Waterfall?).

2.2.1 Extreme programming

There are many advantages with extreme programming (Copeland, 2001 and Wells, 2006). E.g., there will be much less bugs in the software and all members in the group will know exactly how the software is implemented. But there are of course many disadvantages with extreme programming, too. E.g., the working process will be much slower when one person writes code and the other is a spectator or there can be many discussions about everything. They are not necessarily bad; a thoroughly planned project leads to less bugs and more structured code but too much discussions leads to slow downs in the process and of course the fact that the employer pays two salaries for "one" mans job (in a normal employer-employee relationship).

The working methods of this project were;

- Pair programming.
- Weekly meetings, see section 2.2.2.
- Frequent release stages.
- Frequent improvements and design decisions.
- Code standard.
- Collective code ownership.
- Simple design.
- Structured testing.

The above methods characterize the extreme programming methodology.

The above methods do not comprise the full set of extreme programming "functions". Specifically, methods such as;

- Automated testing – as the application did not require automated testing due to it sufficiently small size.

- Official requirement specification – as no such requirement existed from STS or Lund University, LTH.

As methodology from extreme programming was used, we could experiment with our design of the graphical user interface (from here on referenced as GUI) and our backend design during the entire phase 2, something we could never have done if we had chosen e.g. the waterfall model.

From our own experience of extreme programming, we discovered that it's a good methodology to work with. The phases were made together except for Phase 4 where we delegated out pieces, between ourselves, of documentation to write on.

2.2.2 Experiences

Things in a project can always be improved and in our case we didn't document our thoughts in the design phase. That resulted in hours lost to remember how and why we thought a certain way. At the end of phase 2 we decided to make Friday a "document writing" day to speed up the documentation process. This measure should have been done earlier in the project. The documentation process was lacking initially, but with Friday as "document writing" day, the process become natural and the work progressed fine.

Meetings are important in a project. They help the project members be informed about new changes, changed strategy and possible crises. Even if we were just two people that worked on this project, we decided to have meetings in the group every Monday morning to get a briefing of what happened last week, what to do the following week and progress checking. The meetings were kept short (approx. 20-25 minutes) and were very useful to delimitate our project.
Every meeting, we;

- Established what features had to be implemented this week.
- Made a check if we completed last week features.
- Performed risk analysis.
- Assessed time plan and progress.
- Established the work for the following week, broken down by days.
- Discussed other relevant information.

At an early state in our project, we decided to not have an official software, or system, requirement specification.
However, we wrote an unofficial feature list for planned features, where the features were ranked according to their risk, with difficulty and priority as the

axes. STS requested a brief and concise requirement specification, which we delivered. However, the document was rather a feature list, than a normal requirement specification. The document can be seen in Appendix E. Note that the document does not reflect the current state of the application, see chapter 6 for the actual implementation.

The risk analysis was also created in the initial stage of the project, where we listed possible risks within the project. We wrote down the textual risk, the counter-measure, the probability and the degree of consequence, where the probability and consequence degree were the axes of the end risk. The highest ranked risks were;

- Project delimitation – Counter-measure: Frequent meetings between ourselves and STS.
- Time estimation – Counter-measure: Frequent follow ups on the time plan.
- Customer requirement changes – Counter-measure: Frequent meetings with STS.
- Lack of documentation – Counter-measure: Assign work days specifically for documentation and delegate documentation pieces between ourselves.

The only risk that happened that had any affect on the project was the lack of documentation, which was sorted out as explained above.

2.3 Time plan

This project was originally planned for 20 weeks, with 3 milestones, with the deadline 30th of May.

2.3.1 Milestone 1

When we reached this milestone we had succeeded to establish two reference models; one that is based on a real-time strategy point of view and one common for all games.

2.3.2 Milestone 2

Milestone 2 was the middle of phase 2 and indicated that the design had been accepted by our customer. At this point we also knew if we had to delimit part of the project or make some other changes to fit our deadline.

2.3.3 Milestone 3

The last milestone was set to the last week of phase 2. The goal was that the application had to be tested and done for presentation at this point.
The application was to be released as the final product.

2.4 Tools

We used a great deal of tools to accomplish this thesis. The tools listed below are not the only tools, but rather those that were of relevance.

2.4.1 Reference models

We used the following applications to complete the reference models;

- We used *Battlefield* and *World in Conflict* to gather information to respective reference model. The games were arbitrarily selected.
- *Connectivity Designer Edition (CoDE)* is a proprietary information and reference model structuring tool by and for STS. We used this application to create the common reference model, and it is that reference model that will be delivered to STS in CoDE's proprietary format, per their request.

2.4.2 MSDL Scenario Editor

We used the following application to complete our MSDL Scenario Editor;

- The development environment used to create the MSDL Scenario Editor was *Microsoft Visual Studio 2005*.
- The language used to implement MSDL Scenario Editor was *C#* (Hewlett-Packard, Intel, Microsoft, 2008), taking advantage of Visual Studio and the .NET platform, per STS' request.
- As we have implemented the *MSDL specification*, verification with its *XSD-files* (Simulation Interoperability Standards Organization, 2008) and *World Wide Web Consortium's (W3C) XML standard* were required.
- *Google Earth* for OGC-*KML* visualization and utilization.
- *Virtual Battlspace 2* for Virtual Battlespace script file utilization.

2.4.3 Work process

We used the following applications for project process and documentation;

- For a more professional appearance of the tables and overlays in this thesis paper, the applications *XMIND* and *Microsoft Visio* were used.
- *Microsoft Paint* for simple image editing.
- *Microsoft Word* for the documentation process.
- *Microsoft Outlook* for mail correspondence.
- *Microsoft Windows XP* as operating system.

3 Reference models

The first objective was to develop a reference model from a tactical (real-time strategy) game point of view. That means that the starting point is to look from the operational perspective, i.e. the commander's view, and determine what information is most important for the commander and what information shall be sent and retrieved. In the military domain, reference models such as JC3IEDM are used in operational systems and in simulation systems. The new model of MSDL is said to capture all needs for scenario initialization. These reference models have been developed from a military systems operational perspective and not from a gaming perspective. The proposed Strategy Serious Gaming Common Reference Model (SSGCRM) is built from the operational gaming view and the Serious Gaming Common Reference Model (SGCRM) is built from a first-person shooter and an operational gaming view. No comparison has been made in relation to specifications and reference models such as MSDL (SISO, 2008), JC3IEDM (NATO management board, 2007), RPR-FOM (SISO, 1999) or DIS PDUs (SISO, 2005).

The reference models we propose are abstract models of the different types of information one may gather from the SSGCRM and SGCRM. Larger entities may be composed of multiple child entities and may hold overall attributes that the children inherit.

3.1 Strategy Serious Gaming Common Reference Model

RTS games consist of objects that are moved over time in an environment. The player of the game can communicate either with the game and other players and the result of their actions is recorded at a score board. This section is about the Strategy Serious Gaming Common Reference Model that we developed. The SSGCRM was developed by using and observing World in Conflict, by identifying key elements in the interactive entities in the game. See table 3A for an overview.

Strategy Serious gaming common reference model	
	ATTRIBUTES
Objects	Shared properties Static ATTRIBUTES Buildings: Properties Miscellaneous: Properties Dynamic ATTRIBUTES Shared properties Units: Properties Vehicles: Properties
Communication	Properties
Weather	Properties
Map	Properties
Statistics	Properties

Table 3A – Overview of strategy serious gaming common reference model

3.1.1 Objects

Objects are entities that affect game play and can interact with the game play. The Objects entity has two child entities; Static and Dynamic. The children share properties, as can bee seen in Table 3B.

Shared properties of Objects	
	DESCRIPTION
Coordinates	**Coordinates specifying a position on a map**
ID	**Unique identifier**
Name	**Name of the object**
Type	**Type of object**

Table 3B – Shared properties for Objects

3.1.2 Static objects

Static objects are entities that are static throughout a scenario. The Static objects entity has two child entities; Buildings and Dynamic. There are no shared properties between the two children.

3.1.2.1 Buildings

The properties of static objects representing buildings can be seen in Table 3C.

Properties for Buildings	
	DESCRIPTION
Height	Height of the building
Length	Length of the building
Width	Width of the building
Entry points	Entry points of the building

Table 3C – Properties for buildings

3.1.2.2 Miscellaneous

The properties of static objects representing miscellaneous objects in a scenario can be seen in Table 3D.

Properties for Miscellaneous	
	DESCRIPTION
AoE	Area of effect
Trigger	A triggering action based on a condition

Table 3D – Properties for miscellaneous

3.1.3 Dynamic objects

Dynamic objects are entities that are possible to morph and be controlled by users in a scenario. The Dynamic objects entity has two child entities; Units and Vehicles. The children share properties, as can be seen in Table 3E.

Shared properties of Dynamic objects	
	DESCRIPTION
Maximum health	Maximum health an object has
Current health	Current health an object has
Weapon	Type of weapon an object has
Range	Range of the weapon in meters
Damage	Amount of damage an weapon have
Ammunition	Current ammunition an object is carrying
Children	Unique identifiers of the subordinate object
Owner	Unique identifier of the owner for the object
Bot	AI controlled object
Direction	Direction of an object in degrees 0-360
Speed	Current speed in meter per second
State	The current state of the unit. E.g. standing, running etc
Team	What team the object belong to
Rank	Rank of the object
Objectives	Mission objectives
Time	Internal game play time clock
Kills	Amount of killed objects
Deaths	Amount of deaths

Table 3E – Shared properties for dynamic objects

3.1.3.1 Units

The properties of dynamic objects representing units, or humans, can be seen in Table 3F.

Properties for Units	
	DESCRIPTION
Transported	Unique identifier of transportation vehicle, if any

Table 3F – Properties for units

3.1.3.2 Vehicles

The properties of dynamic objects representing vehicles can be seen in Table 3G.

Properties for Vehicles	
	DESCRIPTION
Maximum passengers	Maximum passengers that the vehicle can load
Current passengers	Current passengers in the vehicle
Maximum fuel	Maximum fuel the vehicle can carry
Current fuel	Current fuel in the vehicle
Terrain	Terrain the vehicle is accustomed to

Table 3G – Properties for vehicles

3.1.4 Communication

The communication entity represents communiqués sent and received in a scenario. The properties of the entity can be seen in Table 3H.

Properties for Communication	
	DESCRIPTION
ID	Unique Identifier for the message
ID from	Unique identifier for the originating object
ID to	Unique identifiers for the receiving objects
Message	Message in text

Table 3H – Properties for communication

3.1.5 Weather

The weather entity represents the type of weather in the scenario. The properties of the entity can be seen in Table 3I.

Properties for Weather	
	DESCRIPTION
Type	Type of weather. E.g. snowing, raining etc.
AoE	The area that is affected

Table 3I – Properties for weather

3.1.6 Map

The Map entity represents the type of map the scenario is based on. The properties of the entity can be seen in Table 3J.

Properties for Map	
	DESCRIPTION
Name	Name of the map
Maximum size	Maximum size in x,y,z coordinates

Table 3J – Properties for map

3.1.7 Statistics

The Statistics entity represents general and collective statistics of the mission and its operatives. The properties of the entity can be seen in Table 3K.

Properties for Statistics	
	DESCRIPTION
Total kills	Total kills in the game
Total deaths	Total death in the game
Time played	Total time played in seconds
Current time	Current time since last epoch
Objective	Objectives for this mission

Table 3K – Properties for statistics

3.2 Relationship model for SSGCRM

The relationship model is a diagram of the relations between entities in the reference model and the active operation for each entity relation. Most entities, all of which depicted in Figure 3A, have some form of relation to one another. Each "child" arrow indicates which entity is the child and which is the parent, where "Entities" is the root entity. The nodes marked as “Rel” signify a relationship between two nodes and further the action possible. The arrows indicate in which direction said action(s) can occur. Table 3L explain the cardinality of the nodes marked as “Rel” in Figure 3A.

Cardinality		
	CARDINALITY	CARDINALITY DESCRIPTION
1..1	One-To-One	There may only be one (or zero) entity on either side of the relationship. E.g.: A head and a body have a 1..1 relationship; There may only exist one head for each body and vice versa.
1..n	One-To-Many	There may only be one (or zero) entity on the left side of the relationship and many entities on the right side. E.g.: A person may own multiple cars, but each car only has one owner.
n..n	Many-To-Many	There may be many (or zero) entities on either side of the relationship. Example: A football supporter may support multiple football teams and each football team may have multiple supporters.

Table 3L – Cardinality table

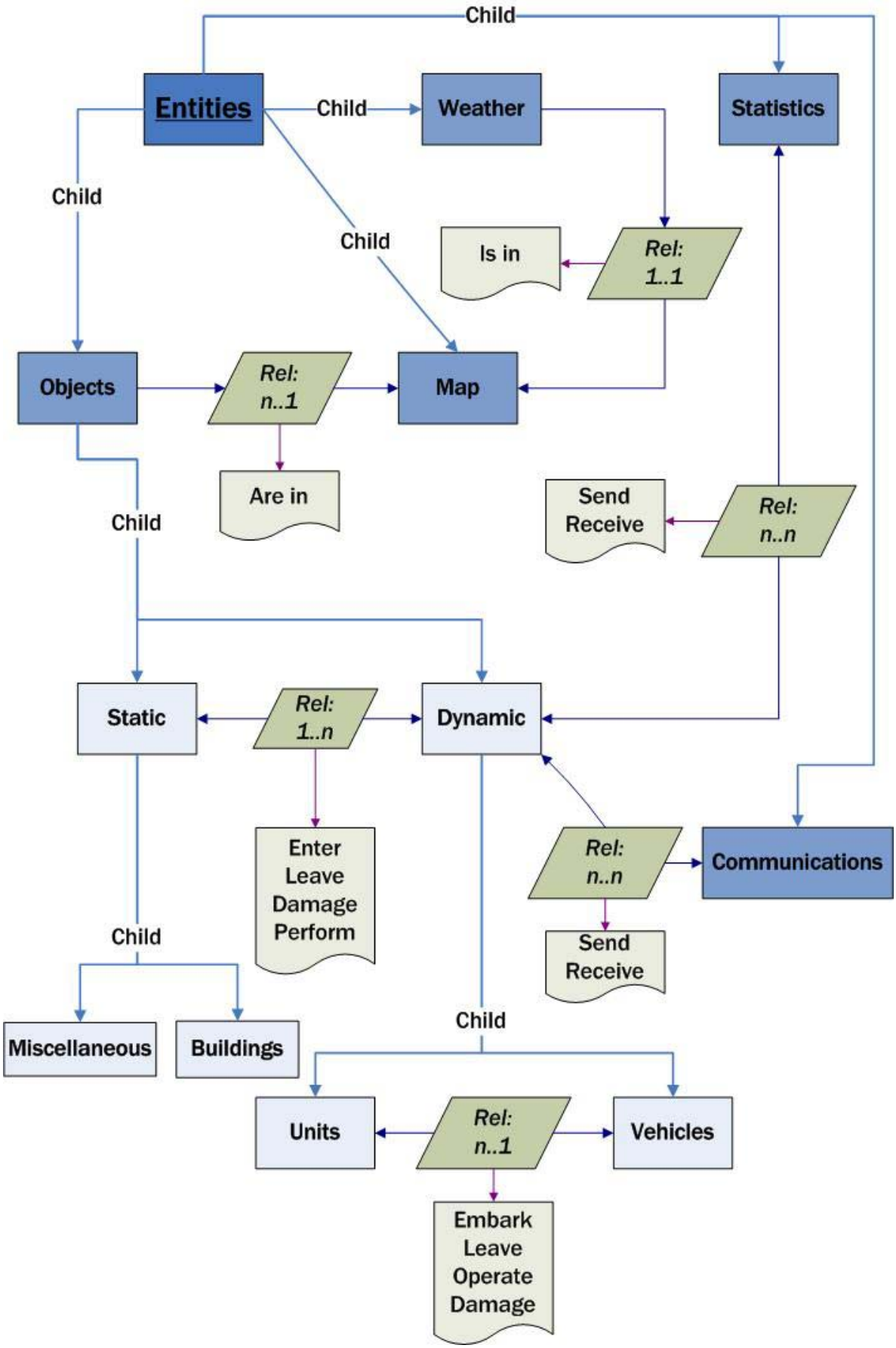

Figure 3A – Relationship model overview

3.2.1 Map – Weather relationship

The Map – Weather relationship can be seen in Figure 3B. The entity relationship is a one-to-one cardinality.

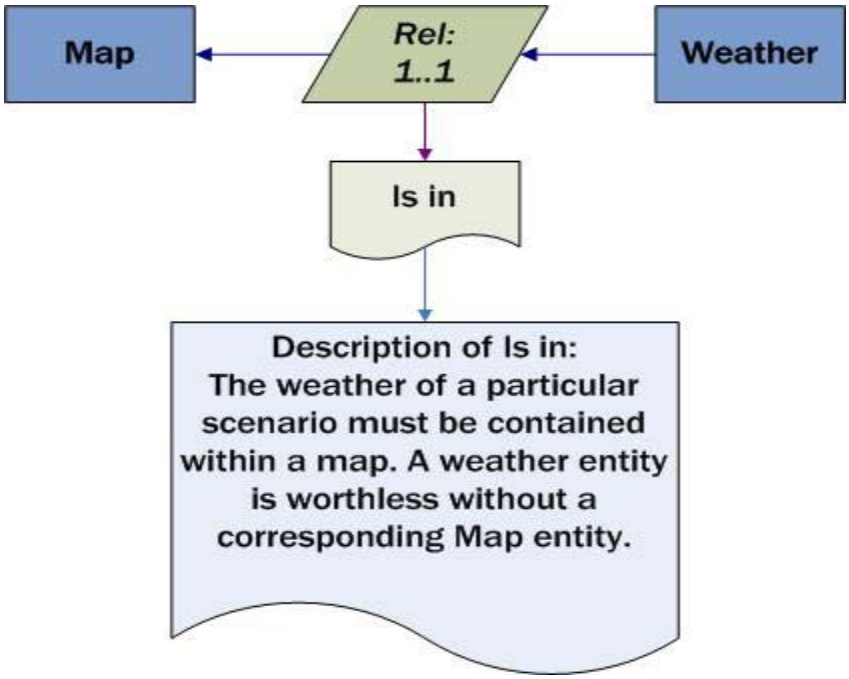

Figure 3B – Relationship of the Map and Weather entities

3.2.2 Map – Objects relationship

The Map – Objects relationship can be seen in Figure 3C. The entity relationship is a one-to-many cardinality.

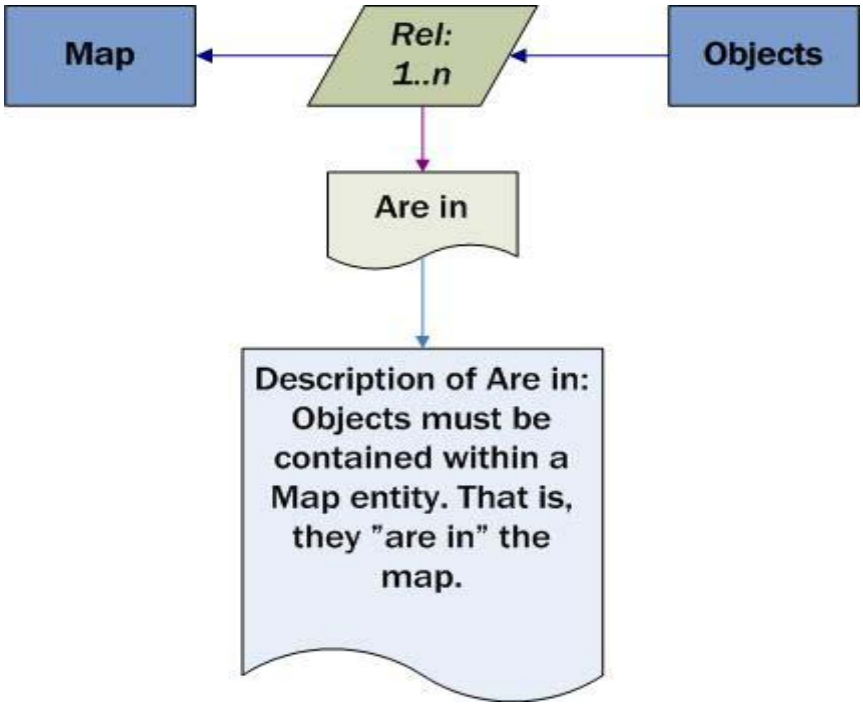

Figure 3C – Relationship of the Map and Objects entities

3.2.3 Dynamic – Statistics relationship

The Dynamic (objects) – Statistics relationship can be seen in Figure 3D. The entity relationship is a many-to-many cardinality.

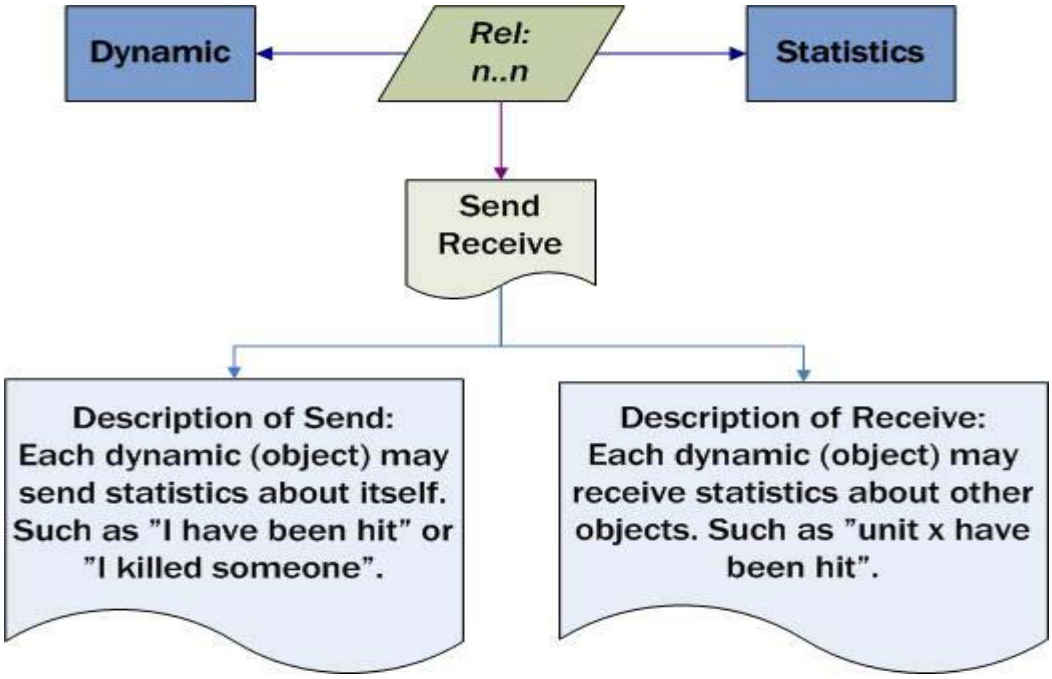

Figure 3D – Relationship of the Dynamic (objects) and Statistics entities

3.2.4 Dynamic – Communications relationship

The Dynamic (objects) – Communications relationship can be seen in Figure 3E. The entity relationship is a many-to-many cardinality.

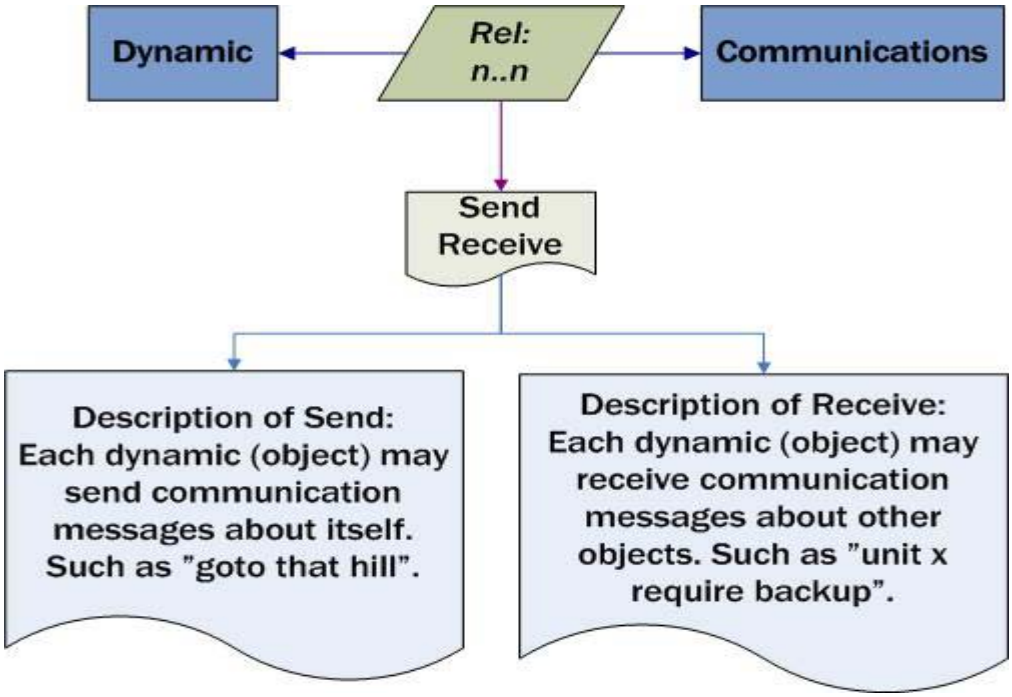

Figure 3E – Relationship of the Dynamic (objects) – Communications entities

3.2.5 Dynamic – Static relationship

The Dynamic (objects) – Static (objects) relationship can be seen in Figure 3F. The entity relationship is a many-to-one cardinality.

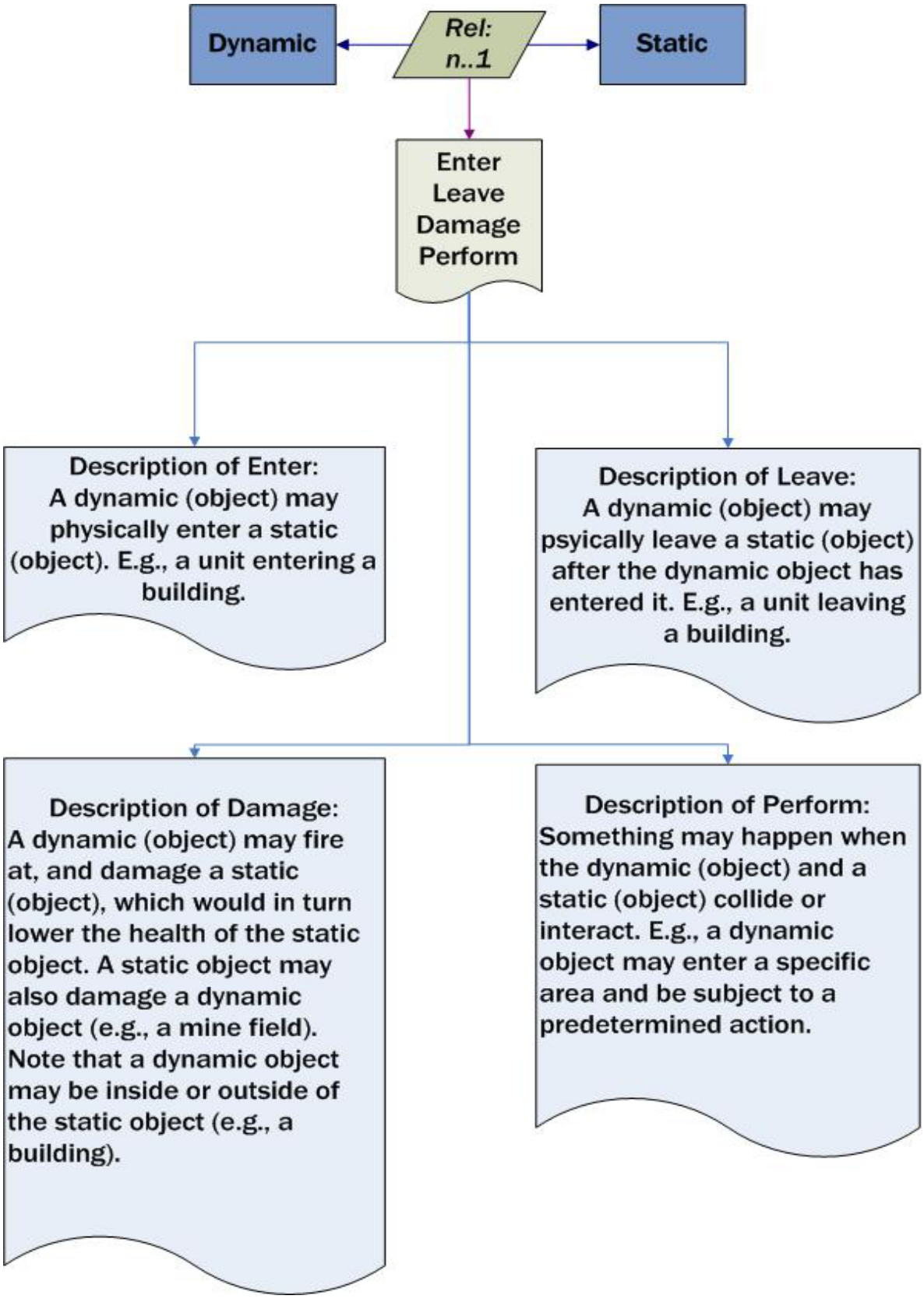

Figure 3F – Relationship of the Dynamic (objects) and Static (objects) entities

3.2.6 Units - Vehicles relationship

The Units – Vehicles relationship can be seen in Figure 3G. The entity relationship is a many-to-one cardinality. Note that Units can perform said actions on other Units, and similarly for Vehicles.

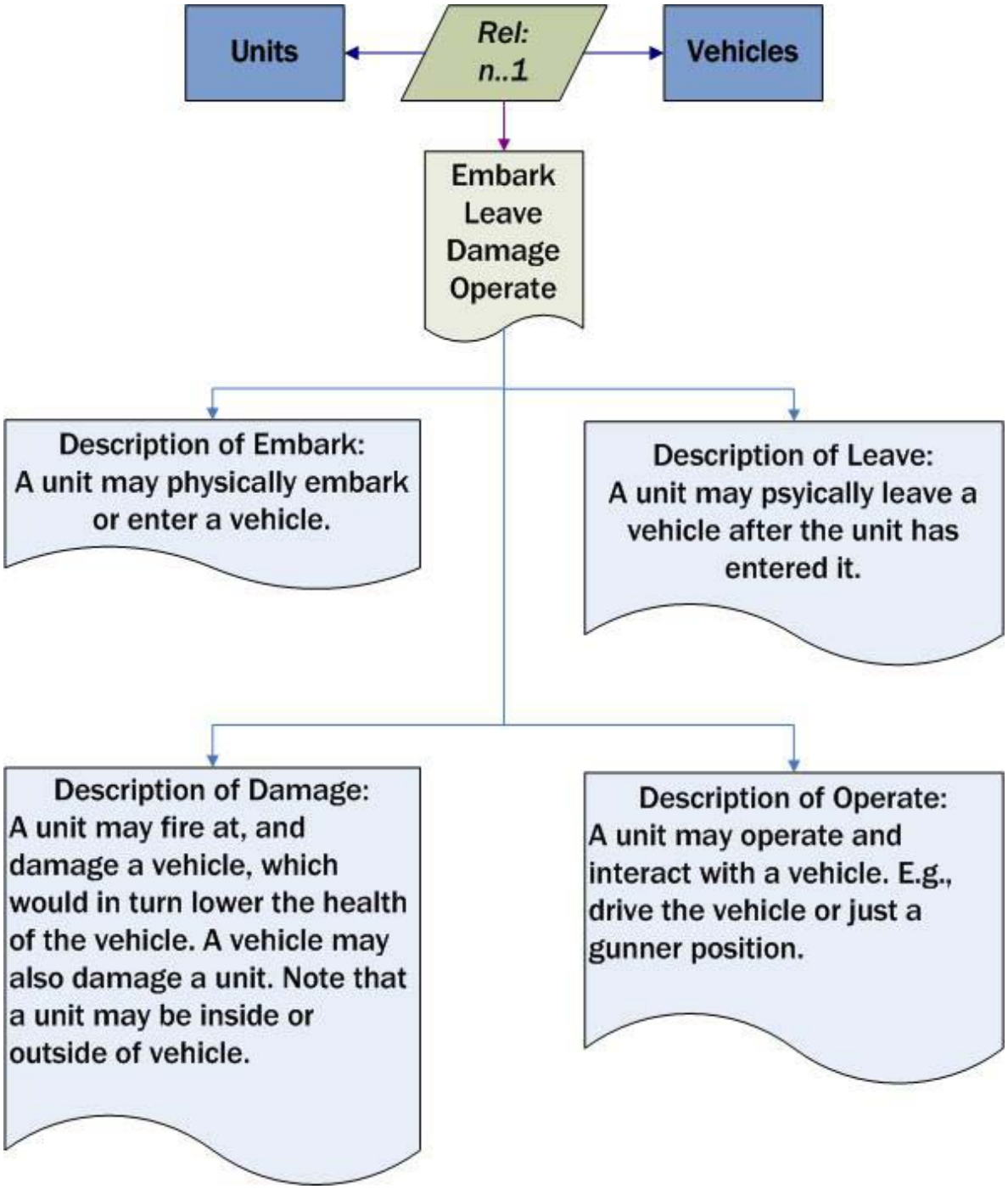

Figure 3G – Relationship of the Units and Vehicles entities

3.3 Serious Gaming Common Reference Model

Serious games consist of objects that are moved over time in an environment. The player of the game can communicate either with the game and other players and the result of their actions is recorded at a score board. This section is about the Serious Gaming Common Reference Model that we developed with the cooperation thesis (Olsson, Michalski, 2008). The SGCRM was developed by using and observing World in Conflict, Battlefield, the SSGCRM and then by identifying key elements in the interactive entities in respective game and reference model. As such, the model may contain information that is not present in the previous reference model in section 3.1 and vice versa. See table 3M for an overview of the reference model and its entities.

Serious gaming common reference model	
	ATTRIBUTES
Weather	Properties
Scenario	Properties
Buildings	Properties
Nature	Properties
Objective	Properties
Units	Properties
Trigger	Properties
Vehicles	Properties
Props	Properties

Table 3M – Overview of the serious gaming common reference model

3.3.1 Weather

The weather entity represents the type of weather in the scenario. The properties of the entity can be seen in Table 3N.

Weather	
	DESCRIPTION
AoE	Area of effect
Type	Type of weather

Table 3N – Properties for the Weather entity

3.3.2 Scenario

The scenario entity contains general and statistic information about the current scenario. The properties of the entity can be seen in Table 3O.

Scenario	
	DESCRIPTION
Deaths	Amount of total deaths
Description	Description about the scenario
Hits	Amount of total hits
ID	Unique identifier for the scenario
Kills	Amount of total kills
Map	Name of the current map
Name	Name of the scenario
Objects	References to all objects in the game
Rounds fired	Amount of total rounds fired
Runtime	How long the game has been in progress
Time of day	Time of day since last epoch

Table 3O – Properties for the Scenario entity

3.3.3 Buildings

The properties of objects representing buildings can be seen in Table 3P.

Buildings	
	DESCRIPTION
Current health	Current health of the building
Dimensions	Coordinates of building model
Direction	Direction the building is facing
ID	Unique identifier
Maximum health	Maximum health of the building
Name	Name of the building
Position	The coordinate of the building's center
Type	Type of building

Table 3P – Properties for the Buildings entity

3.3.4 Nature

The properties of objects representing nature elements can be seen in Table 3Q.

Nature	
	DESCRIPTION
Current health	Current health of the nature element
Dimensions	Shape of the model
Direction	Direction the nature element is facing
ID	Unique identifier
Maximum health	Maximum health of the nature element
Name	Name of the nature element
Position	The coordinate of the nature element's center
Type	Type of nature element

Table 3Q – Properties for the Nature entity

3.3.5 Objective

The properties of the entity representing an Objective can be seen in Table 3R.

Objective	
	DESCRIPTION
ID	Unique identifier of the objective
Priority	The priority of the objective
Recipients	The unique identifiers of the recieving objects
Source	The unique identifier of the initiating object
State	The state the objective is in. E.g., in progress or completed
Timeout	How long the objective is valid
Description	Description of the objective

Table 3R – Properties for the Objective entity

3.3.6 Trigger

The properties of the entity representing a Trigger can be seen in Table 3S.

Trigger	
	DESCRIPTION
AoE	Area of effect
Type	Condition to engage trigger
Action	List of actions to be executed

Table 3S – Properties for the Trigger entity

3.3.7 Props

The properties the entity Props representing miscellaneous items can be seen in Table 3T.

Props	
	DESCRIPTION
Current health	Current health of the prop
Dimensions	Shape of the model
Direction	Direction the prop is facing
ID	Unique identifier
Maximum health	Maximum health of the prop
Name	Name of the prop
Position	The coordinate of the prop's center
Type	Type of prop

Table 3T – Properties for the Props entity

3.3.8 Vehicles

The properties of objects representing vehicles can be seen in Table 3U.

Vehicles	
	DESCRIPTION
Ammo	Ammunition left for the current weapon
Bot	Human or computer AI
Commander	The commanding unit or vehicle
Current fuel	Current fuel of the vehicle
Current health	Current health of the vehicle
Current passengers	Current passengers in the vehicle
Deaths	Amount of deaths
Equipment	List of equipment the vehicle is carrying
Hits	Amount of hits
Hit zones	Vehicle's hit zones
ID	Unique idenifier
Kills	Amount of kills
Maximum fuel	Maximum fuel in the vehicle
Maximum passengers	Maximum passengers the vehicle can carry
Maximum health	Maximum health of the vehicle
Name	Name of the vehicle
Objectives	List of objective IDs the vehicle is supposed to pursue
Position	The position of the vehicle
Rank	Rank of the vehicle
Rounds fired	Amount of rounds fired
Speed	Current speed
State	Current state of the vehicle. E.g., transporting, aiming etc
Subordinates	Vehicles or units that this vehicle can command
Target hits	Amount of target hits
Team	The team the vehicle belongs to
Time	Internal game play time clock
Transport vehicle	Vehicle the vehicle is transported in
Vehicle type	The vehicle's terrain speciality
View direction	The direction of the vehicle
Weapon	List of weapons the the vehicle is carrying
Weapon range	Range of the current weapon

Table 3U – Properties for the Vehicles entity

3.3.9 Units

The properties of objects representing units, or humans, can be seen in Table 3V.

Units	
	DESCRIPTION
Ammo	Ammunition left for the current weapon
Bot	Human or computer AI
Commander	The commanding unit
Current health	Current health of the unit
Deaths	Amount of deaths
Equipment	List of equipment the unit is carrying
Hits	Amount of hits
Hit zones	Unit's hit zones
ID	Unique idenifier
Kills	Amount of kills
Maximum health	Maximum health of the unit
Name	Name of the unit
Objectives	List of objective IDs the unit is supposed to pursue
Position	The position of the unit
Rank	Rank of the unit
Rounds fired	Amount of rounds fired
Speed	Current speed
State	Current state of the unit. E.g., atanding, running etc
Subordinates	Units that this unit can command
Target hits	Amount of target hits
Team	The team the unit belongs to
Time	Internal game play time clock
Transport vehicle	Vehicle the unit is transported in
View direction	The direction of the unit
Weapon	List of weapons the the unit is carrying
Weapon range	Range of the current weapon

Table 3V – Properties for the Units entity

3.4 Relationship model for SGCRM

The following relationship model reflects the common reference model in the previous section. The entities can be seen in figure 3H. For further information about the interpretation of the relationship model, see the introduction to section 3.2.
The blocks depicted in figure 3H describe a union between the entities inside of said block. There are three blocks;

- Static objects block – Consists of the entities “Nature”, “Props” and “Buildings”.
- Dynamic objects block – Consists of the entities “Units” and “Vehicles”.
- Objects block – Consists of the blocks “Static objects” and “Dynamic objects”, and the entities “Weather” and “Trigger”.

The blocks are only for illustrative purposes.

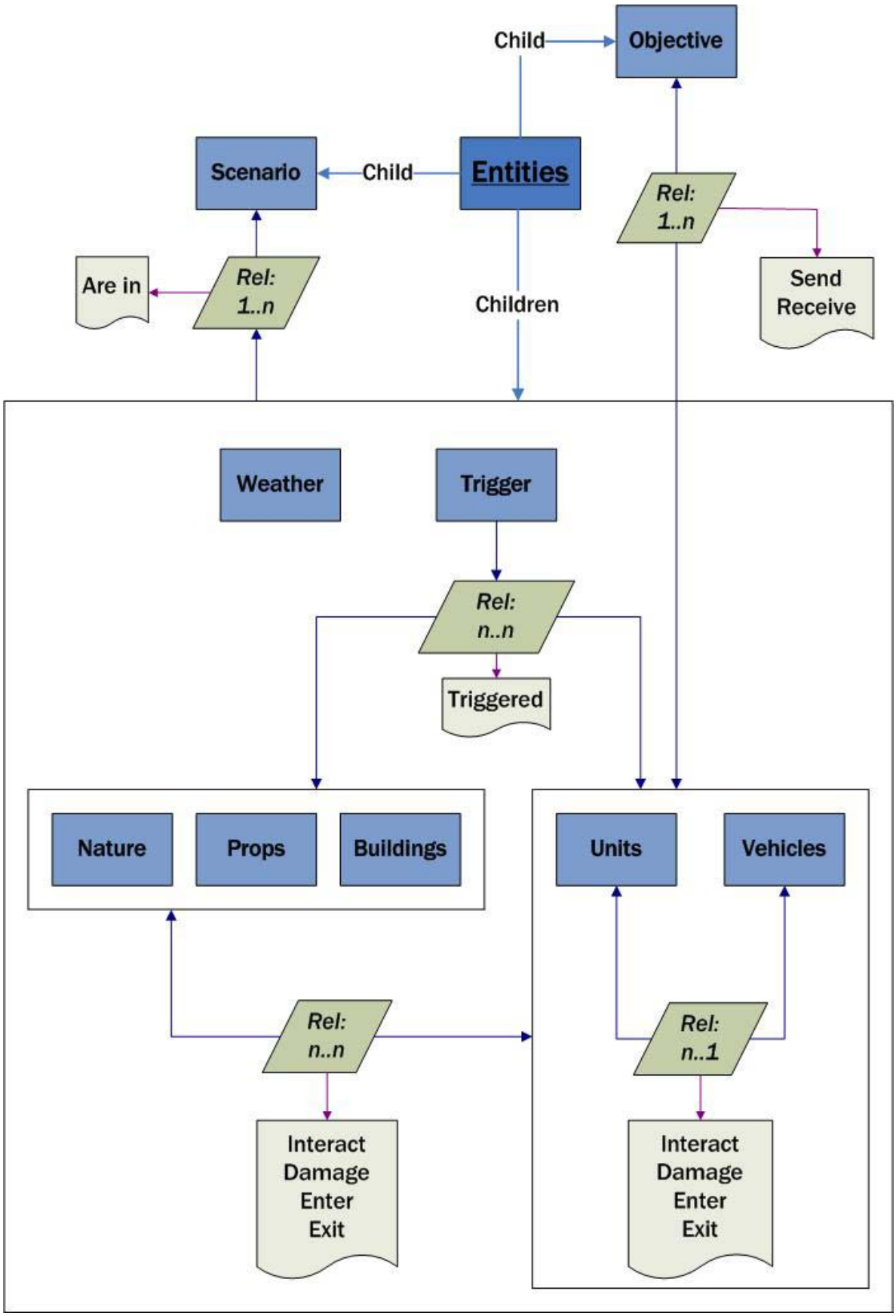

Figure 3H – Relationship model overview

3.4.1 Units – Vehicles relationship

The Units – Vehicles relationship can be seen in Figure 3I. The entity relationship is a many-to-one cardinality. Note that Units can perform said actions on other Units, and similarly for Vehicles.

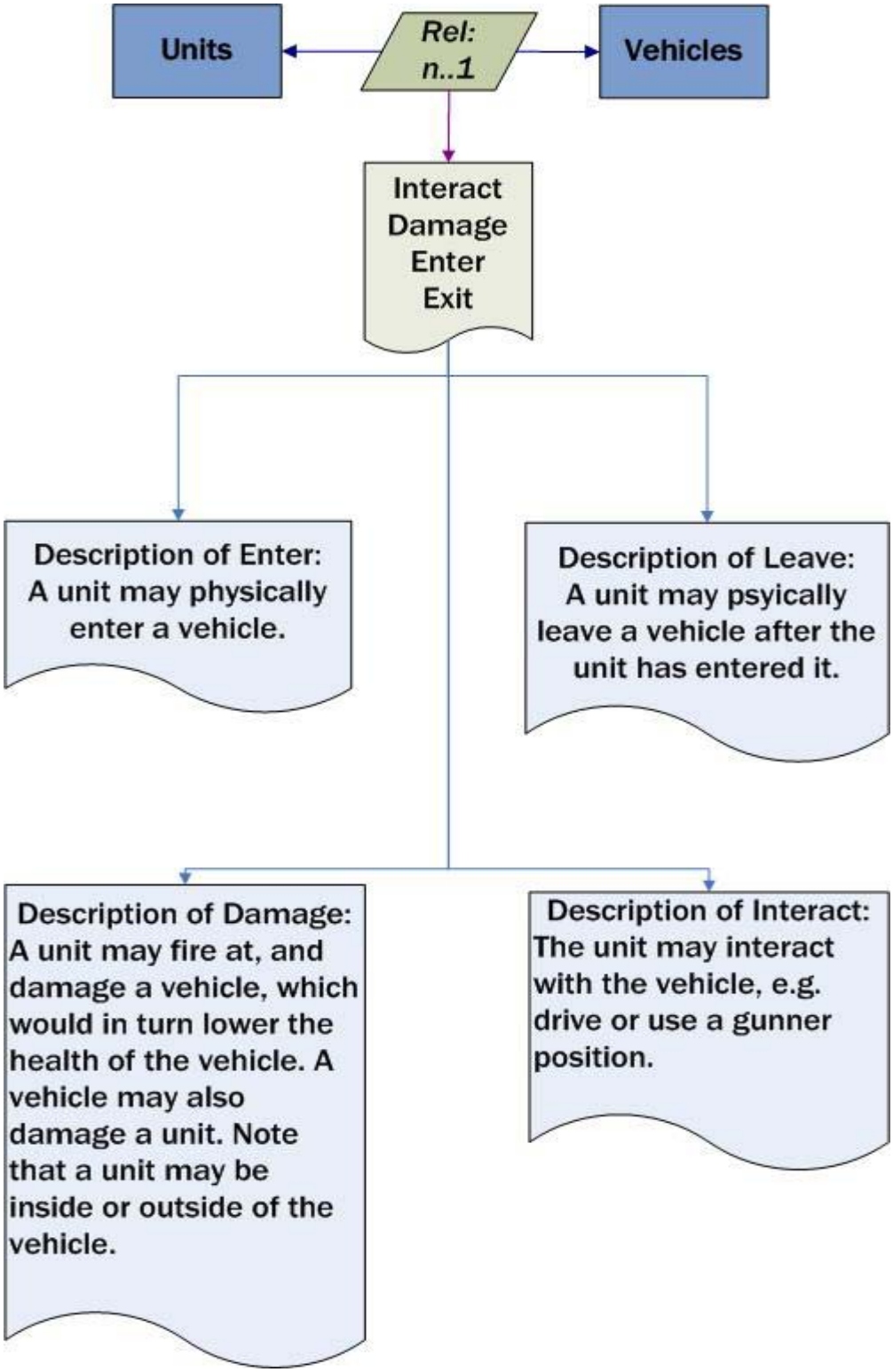

Figure 3I – Relationship of the Units and Vehicles entities

3.4.2 Dynamic objects block – Static objects block relationship

The Dynamic objects block – Static objects block relationship can be seen in Figure 3J. The block relationship is a many-to-many cardinality.

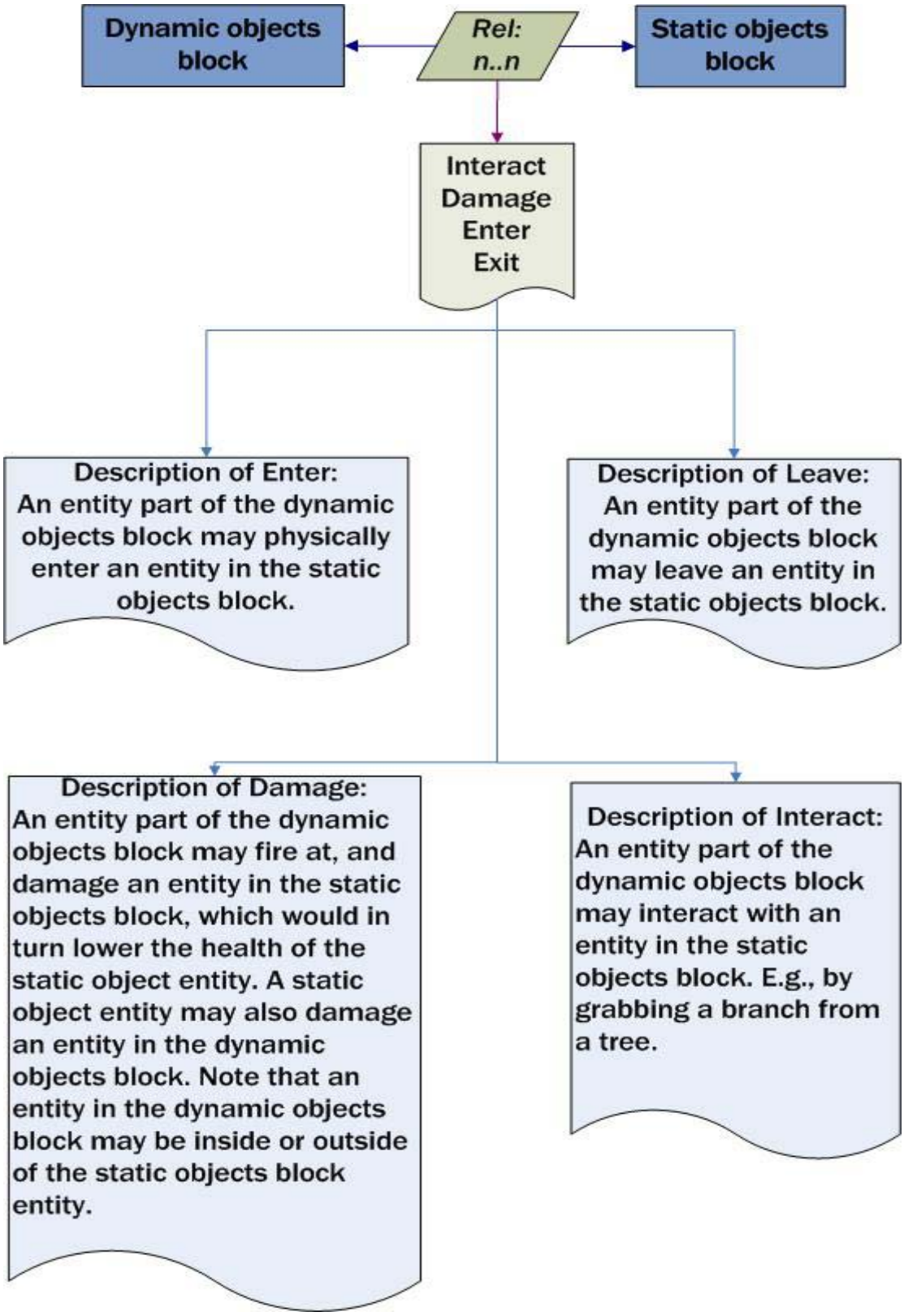

Figure 3J – Relationship of the Dynamic objects and Static objects block

3.4.3 Dynamic objects block – Static objects block – Trigger relationship

The Dynamic objects block – Static objects block – Trigger relationship can be seen in Figure 3K. The block and entity relationship is a many-to-many cardinality.

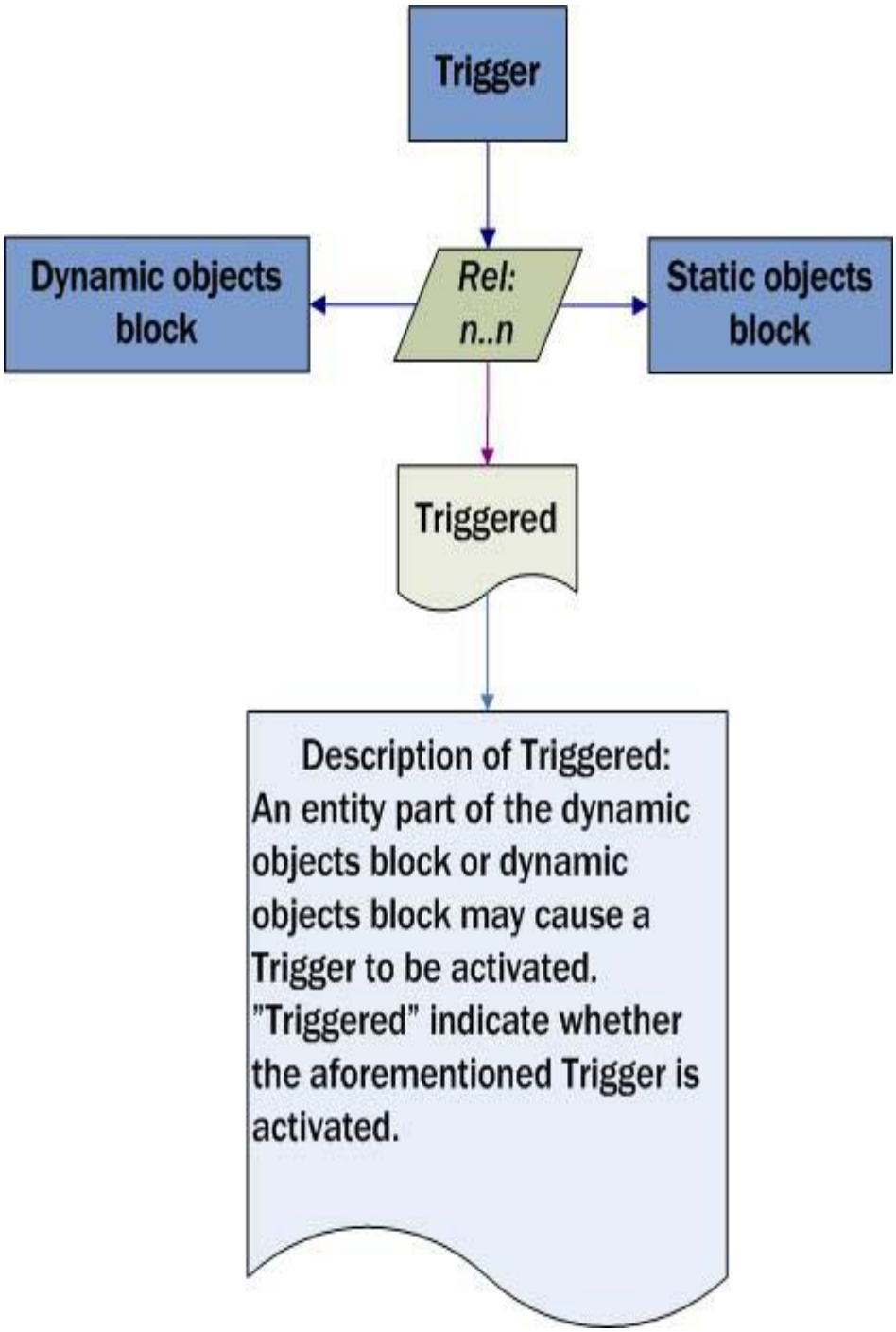

Figure 3K – Relationship of the Dynamic objects and Static objects block and the Trigger entity.

3.4.4 Dynamic objects block – Objective relationship

The Dynamic objects block – Objective relationship can be seen in Figure 3L. The block and entity relationship is a many-to-one cardinality.

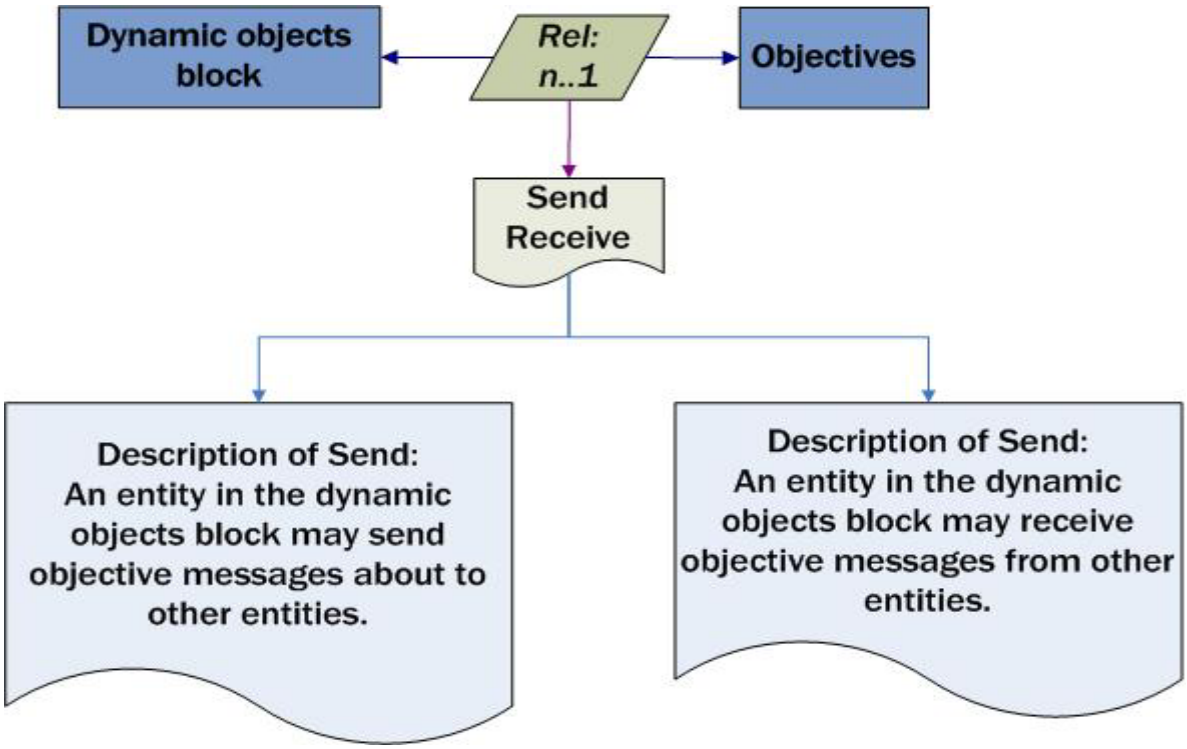

Figure 3L – Relationship of the Dynamic objects block and the Objective entity.

3.4.5 Scenario – Objects block relationship

The Scenario – Objects block relationship can be seen in Figure 3M. The entity and block relationship is a one-to-many cardinality.

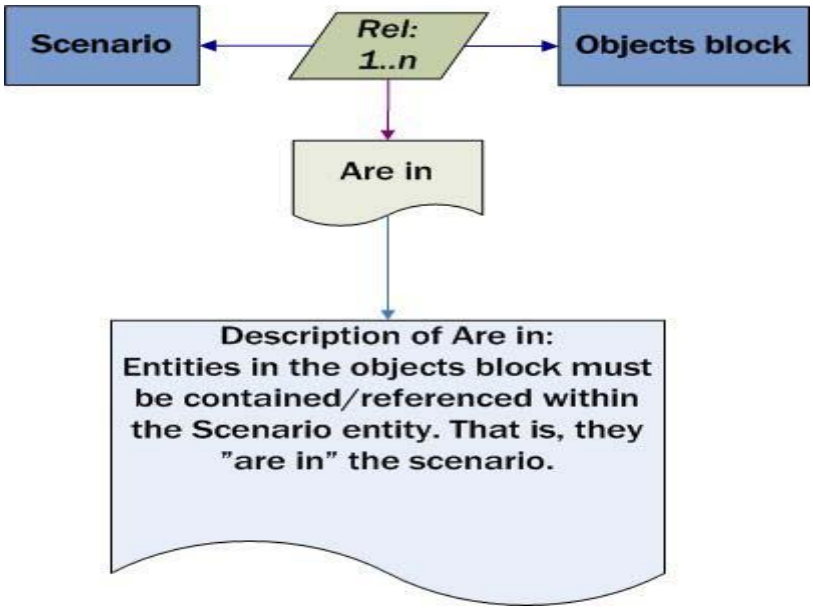

Figure 3M – Relationship of the Scenario entity and Objects block.

4 Military Scenario Definition Language

The Military Scenario Definition Language (SISO, 2008) is a standard created by SISO. Its intent is to provide a standardized way for loading and using militarized scenarios and simulations. It will from now on be referenced as "the specification", unless noted. The specification defines a scenario in an XML (World Wide Web Consortium, XML standard) schema, allowing total or part of scenarios to be used within one simulation. A MSDL scenario is only a snapshot of a simulation; the course of action and the context of the MSDL scenario make up the situation.

The specification requires and uses several standards and concepts, see Table 4A.

Standards and concepts	
	USAGE IN MSDL
MIL-STS-2525B Common Warfighting Symbology	Symbol references for units, equipment etc.
Joint Command, Controls and Consultation Information Exchange Data Model (JC3IEDM)	Joint command references and connections.
Military operations other than war (MOOTW)	Graphical symbol references.

Table 4A – Standards and concepts

4.1 Current specification

4.1.1 General

The specification is currently at version 1.0 (SISO, 2008).
The specification is based on an XML schema structure; a master node with children, where the children themselves may have children, and so on.
Verification of a scenario can be performed with accompanying XSD-files (SISO, 2008). As well, the specification document is created through the XSD-files.
Each node may have attributes, regardless if there are child nodes.
A node can be mandatory or optional.

There are four namespaces, or rather prefixes, to which nodes belong to; "msdl", "modelID", "jc3iedm" and "mootw". A namespace is "a scoping construct to subdivide the set of names and their visibility within a system" (Lea, 1995).
The msdl namespace is the most commonly used in the specification.

The structure of each node's children is defined in the specification, section 5.3.3.5 Compositors, as sequence, choice or all; if a node is marked as sequence, all children must come in that exact sequence; if a node is marked as choice, there can only exist, at most, one child; if a node is marked as all, all children may be present, but regardless of order.
The number of children or attributes is specified, and can be zero or one, bound between an interval, and unbounded between a starting position and "infinity".
An attribute's values are restricted to the attribute's data type. Chapter 7, in the specification, has a list of those data types.

The root node in a scenario is called "MilitaryScenario", which is an "all composite". See Table 4B for brief summary of its children. An example of a scenario file without units can be seen in Appendix A and a brief explanation of the file in section 4.3. An example of a scenario file with a unit can be seen in Appendix B and a brief explanation of the file in section 4.4.

Child nodes of MilitaryScenario

	DESCRIPTION	MANDATORY
ScenarioID	Scenario identifier.	Yes
Options	MSDL and scenario options.	Yes
Environment	The scenario's terrain and environment.	No
ForceSides	The forces and sides of the scenario.	Yes
Organizations	Scenario structure of the organizations and equipment.	No
Overlays	Logical overlays for groups of intelligence.	No
Installations	Describe the detected installations from the intelligence gathered.	No
TacticalGraphics	Tactical information known about a force or unit.	No
MOOTWGraphics	Information about the graphically detected military operations, other than war.	No

Table 4B – Child nodes of MilitaryScenario

4.1.2 Issues
As the specification is laid out as a tree, it is possible to read and use only part of the specification, without affecting all other branches, or nodes.

However, certain nodes require references to other nodes, which are done by unique identification links. While there may not be a reference relation between two nodes, in the form of a unique identification, a node may require the presence of another to exist, depending on node mandating rules.
Both situations can be troublesome, specifically in the former; if a node is removed, what happens with the nodes that reference said node? Currently, there's no solution in the specification. The only solution to the second problem would be to make all nodes optional, which we don't think is acceptable.

The MSDL format requires that all mandatory nodes must be present, regardless of the intended message with the MSDL scenario. E.g., say if an application shall import a scenario containing 5000 units and later add one unit. The application's logic must either accept adding that one unit to an existing scenario or the entire scenario with 5001 units must be created and loaded. The former situation is of course desired, but also the more difficult to programmatically achieve.

Two substantial issues with the current specification is the lack of localization and notification about which encoding the final MSDL XML shall use. These issues will not be troublesome for MSDL users that use the same language or encoding, but it will be difficult when multiple users of various countries and cultures shall interact. In fact, rendition and generation software may break or behave unexpectedly if they aren't aware of encoding.
We recommend using the standardized encoding UTF-8 (ISO, 2003) for international use.

The specification's use of symbology for units and equipment is very simple; all units and equipment is tracked by their symbol identifier, "SymbolID", which is a MIL-STD-2525B symbol identifier. Unfortunately, this also means that a unit node's children does not automatically determine which type of unit it is, but the unit's symbol identifier. We're unsure whether this can or ought to be mandated through the XSD schema files or through the specification, but the former would allow more automation in applications. This confusion may also vary depending on rendition software; one application may ignore the symbol identifier and parse the child nodes, while another application may ignore the child nodes and strictly follow the symbol identifier.
A severe issue with the MIL-STD-2525B symbology in the specification is that certain characters are explicitly omitted when used; the second and forth characters; a unit's force and the unit's status regarding presence. While the forth character does not have any relevance for scenario generation, a unit's force is very relevant for scenario import. E.g., when a scenario editor imports a scenario, it cannot know, by a unit's symbol identifier, which force the unit

belongs to. By force, we mean "friend", "hostile" etc from the MIL-STD-2525B standard. As such, the scenario editor must guess (whether it guesses well or poorly is irrelevant) which force is used and apply that symbol in the scenario editor's view.

Equipment items in a scenario always have a link to the unit that owns the equipment item. However, the equipment item’s symbol identifier may indicate that the equipment item isn’t technically equipment. Instead, the symbol identifier may be a normal ground unit, even a tank or aircraft.

While the documentation of a unit's and equipment’s speed is concise, the specification does not restrict speed in regards to its base type. That is, a normal ground unit (person) may have a higher speed than a tank, which of course isn't possible in the real world. As well, the speed of water or subsurface units and equipment is not specified.

There are simulations where units’ health condition is an important key factor. E.g., in a rescue simulation, where the hostage is injured or incapacitated. However, there is no health indicator in the specification. Moreover, security vests or other security devices for the health of the unit do not exist in the specification.
Additionally, equipment items – weapons – have no actual damage attribute. As such, it is impossible to calculate damage done to units.

The node “SupportRelation” have an attribute “SupportType” that indicate the type of support a unit gives to its supporting unit. The “SupportType” attribute is an enumeration, consisting of a set of values, among them “GS-R”. Most common computer languages such as C++, Java and C# offer enumeration-capabilities to the programmer, allowing above structures to be specified more elegantly. However, the “GS-R” value is troublesome; it is not possible to use the character “-” in any of the above languages’ enumerations. Additionally, the dash is not, to our knowledge, used elsewhere in the specification, and we don’t think it should be present here either.

4.1.3 Flexibility

The specification allow much flexibility, specifically because it is implemented using XML. The use of XML allows the MSDL structure to be extended, without interference from the continuing effort to evolve the base specification.

A flexible area is the specification’s use of coordinate system, where four different systems may be used; Military Grid Reference System (MGRS) (Lampinen, 2001), Universal Transverse Mercator Coordinate (UTM)

(Lampinen, 2001), geodetic coordinates (GDC) (Anon, Geodetic and geocentric coordinates) and geocentric coordinates (GCC) (Anon, Geodetic and geocentric coordinates). It is allowed to mix and match different coordinate systems within one MSDL XML file, however confusing that may be for end users or applications.
While coordinate system flexibility is fine, for units and equipment, there is no requirement for a coordinate system for those items. The lack of requirement can cause a major difficulty for importing scenarios; where should the application place a unit (graphically) if the unit has no coordinate? Additionally, a unit (or its equipment) may have a coordinate that is not confined in the area specified in the "Environment" node (which specifies the "scenario area").
An odd issue with the GDC implementation in MSDL is that the range is specified to -179 to 180, while this is incorrect. The correct range is -180 to 180 and ought to be fixed in a future revision.
Normally, computer screen manufacturers count the top left corner of the screen as the 0.0 point. However, MSDL count the top right corner of the screen as the 0.0 point, which we believe is odd.
As is noted in section 6.4.1, we have not implemented a coordinate system other than geodetic coordinates. As such, we have no comment on the other coordinate systems.

While flexibility is always good, a unit can have a very strange mixture of properties, all attributed to MSDL's flexibility. For example, a unit can be represented by its symbol identifier as a ground unit; the unit's coordinate may indicate that it is in the air; a formation type as sub surface and; formationdata as a surface formation. In advanced MSDL rendering software, this flexibility may cause issues for visual representation, as the information (mostly) contradicts each other.

The specification supports a large portion of, if not the entire, JC3IEDM concept. The specification also requires the support of the full MIL-STD-2525B. While the mandated support is, in themselves, fine for JC3IEDM and MIL-STD-2525B, we believe focus is lacking for units and their properties. E.g., more properties ought to be added for a unit to indicate whether it's a tank or a humvee, and not depend on the unit's MIL-STD-2525B representation.

4.1.4 Existing solutions

Beside the implementation presented in this paper, the only available commercial software using MSDL is One Semi-Automated Forces (OneSAF), which is a program suite made up of simulation and modeling editors. We do not know the extent of OneSAF's MSDL implementation, other than that it is

implemented using a plug-in with Microsoft PowerPoint (OneSAF, OneSAF's MSDL implementation).

4.2 Future revisions of MSDL

The specification is continuously updated and worked on. The future specification will possibly (Anon, 2008, Future MSDL focus) bring the following enhancements;

- Loosen the specification's tie with doctrine specific to U.S.A.
- Addition of Coalition-Battle Management Language (C-BML), allowing mission specific information to be sent. E.g., an order or plan from a superior commander may consist of where a unit should go, what the unit shall do, when the unit shall perform the action, why the unit shall do it, and who, specifically, shall do it.
- Modeling of possible threats, in conjunction with the MIL-STD-2525B standard.
- Use of type and instance enumerations to distinguish units, and the use of multiple standards to achieve this combination.
- Further extension of existing MSDL components.

It is possible to join the development group and be part of future MSDL discussions.

See section 8.2 for MSDL suggestions.

4.3 Brief explanation of Appendix A

As noted in 4.1.1, a scenario is required to start with "MilitaryScenario". The "xmlns" tags are only used for XML verification against XSD schema files and may be omitted if the rending application is able to handle it.
The children of MilitaryScenario are;

- "ScenarioID"
- "Options"
- "Environment"
- "ForceSides"

See table 4B for respective general description.

ScenarioID's attributes are in the "modelID" namespace, which is the reason "modelID" is prefixing the attributes, while all other attributes are in the "msdl" namespace, and do not require a prefix.

The MSDL version used in the file is "1.0", as described in the attribute "MSDLVersion".

The scenario file illustrates a map, with GDC coordinates. The GDC coordinates are 180, 180 for the top right coordinate and -179, -179 for the bottom right coordinate. This illustrates a map representing the entire world.

The scenario file only contains one force, called “Civilian”. A unit must not exist for a force to exist, as the ForceSides node is mandatory.

For more information about nodes and their attributes, see respective section in the specification.

4.4 Brief explanation of Appendix B

The scenario file in Appendix B is an extended version of the scenario file in Appendix A, with a unit added.

The unit created have a MIL-STD-2525B symbol of “S-G----------“, which is basically only a “ground unit”.

The GDC coordinate the unit have is 10.3678, 61.18505 which is somewhere on the west coast of Norway.

The unit’s referenced force is the force described in the previous section.

For more information about nodes and their attributes, see respective section in the specification.

5 Design of the MSDL Scenario Editor

The initial purpose of the MSDL Scenario Editor (from here on referenced as MSDLSE) was to create a proof-of-concept application that implemented, in part, the MSDL specification and allowed a user to generate arbitrary scenarios. As such, the application had to have three parts;

- Graphical user interface (GUI).
- Back end.
- Scenario module.

The above brake down in parts is also the current design, as described further in this chapter.

The summary of the communication flows between the parts are illustrated in figure 5A, 5B and 5C.

In figure 5A, the communication flow only involves the GUI and back end.

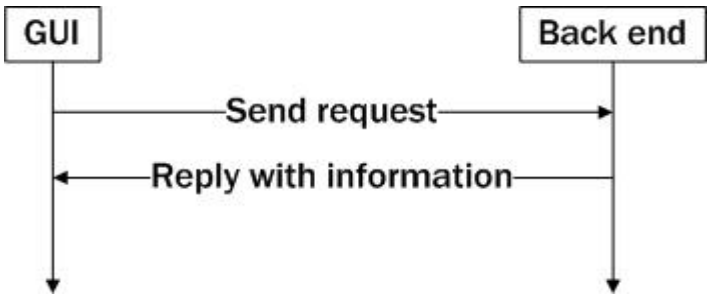

Figure 5A – GUI and back end communication flow

In figure 5B, is the scenario module is part of the communication flow. Further, in this illustration the scenario module does not send back any information and neither does the back end to the GUI. As such, the GUI will not anticipate information.

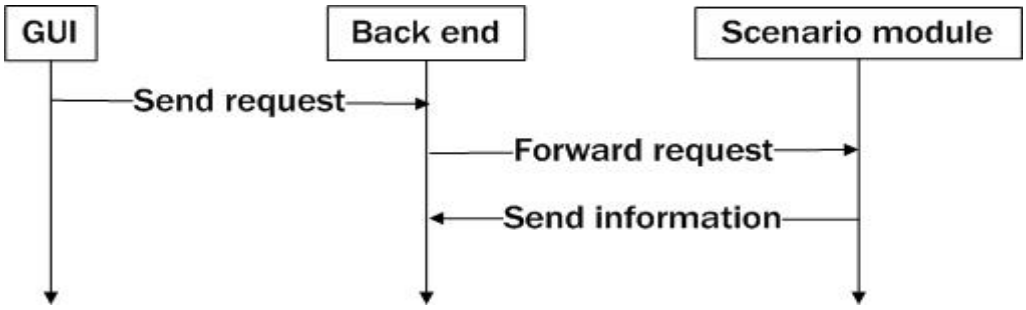

Figure 5B – GUI, back end and scenario module communication flow

Contrary to figure 5B, figure 5C illustrates the flow when the scenario module does send information to the back end. Usually, though not always, will the back end forward the information to the GUI, if the GUI requested such information.

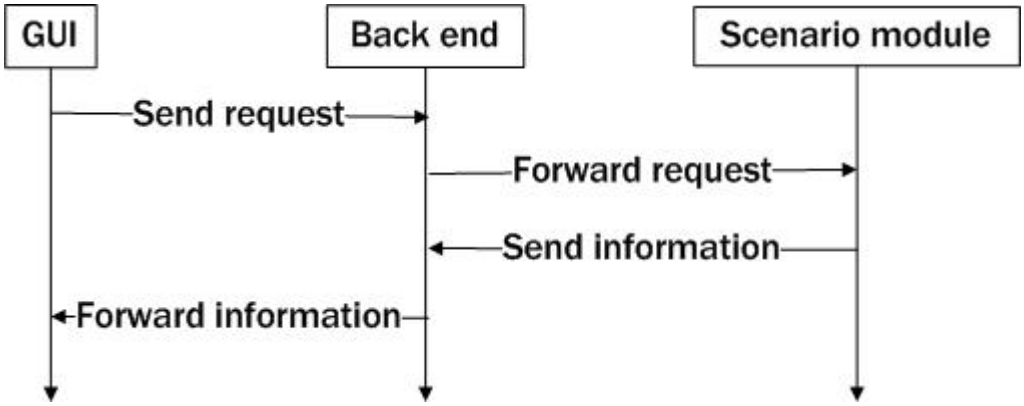

Figure 5C – GUI, back end and scenario module communication flow

For detailed information on the implementation and features of MSDLSE, see chapter 6.

5.1 Graphical user interface

To allow a wide range of users, specifically non computer savvy users, the user interface has to be graphical and easy to use.
The interface is in charge of the interaction between the user and the rest of the application; the user must not see or notice other system parts.

The design of the user interface can be divided in two sections; programmatic design and interactive design for the user.

5.1.1 Interactive design

The interaction in MSDLSE is broken down by the following sections;

- Menu interaction – Items in the File, Edit and Help menu.
- Tabbing interface – Selection of work space.
- Tool box view – Selection of tools available for the work space.
- MSDL component view – Selection of MSDL components.
- Single and multiple property view for MSDL components – Selection and editing of MSDL component properties.
- Satellite image selection – Selection of source satellite images.
- MIL-STD-2525B symbol pallet – Selection of MIL-STD-2525B component.
- Combined satellite image and MIL-STD-2525B management – Interacting with a work space's satellite image and its components.
- Settings view – Editing of settings for the application and project.

- Keyboard interaction – Shortcut commands for the user.

The above sections allow combinations of different types of management, per the previous section.

The satellite image selection allows the user to change the source of the satellite image, regardless if the image is localized or on a network.

The selection of the MIL-STD-2525B symbols is dynamically generated, allowing the user to use multiple different MIL-STD-2525B symbol sets.

5.1.2 Programmatic design

MSDLSE's GUI is programmatically broken down by area of interest for the user. That is, the GUI separates several different concepts;

- Project management – A project is composed of multiple scenarios.
- Scenario management – A scenario is composed of a satellite image and a MSDL scenario tree.
- Settings management – Settings for the application as a whole and as well the project's settings.
- Interface controls management – Controls are interactive components for the user.
- Interface control event management – Each control may have events; actions that are performed when a precondition is met.
- Satellite image management – Interaction between the user and the work space's satellite image and as well the import of satellite images.
- MIL-STD-2525B symbol management – Interaction between the user and the work space's MIL-STD-2525B components and as well MIL-STD-2525B component selection.

The GUI also utilizes the following external components;

- MapDraw, provided by STS.
- Splash screen, provided by STS.
- About box, provided by STS.
- SharpMap (Nielsen, 2008).

MapDraw's purpose is to allow arbitrary image drawing and image calibration according to specified system coordinates.
The splash screen is a popup screen during MSDLSE's initiation, displaying the STS logo type.
The about box is a popup screen displaying version number and copyright for MSDLSE.
SharpMap's purpose is to allow rendering of vector graphic images (.shp).

5.2 Back end

The back end of MSDLSE is the engine that "drives the locomotive". The back end manages;

- Forwarding of requests by the GUI to appropriate scenario module.
- Forwarding of acknowledgements and information by the scenario module to the GUI.
- Project storage and management.
- Logging information and errors in the application.
- Settings for projects and the application as a whole.
- Common and general functionality for classes in MSDLSE.

The back end serves as a component for scenarios to forward requests by the GUI and scenario module(s), and vice versa.

The logging facility enables writing to a file for errors and for information messages. If needed, it would be possible to write elsewhere, too, with minimum trouble.

The management of the settings allow users to, at runtime, change the project work space and as well MSDLSE's settings, beyond the project.

The "common and general functionality for classes" is functionality that allows programmatically simpler structuring and usage of code. E.g., code and information that is identical in multiple classes or methods shall reside in one place. Therefore, any required changes are only applied once; instead of having to copy the identical information in multiple places. Additionally, the functionality is not scenario module dependant, allowing further development without other code modification.

The initial design of MSDLSE had no concept of "project". Instead, scenarios were kept in the notion of a "database". However, due to similarities in of the "database" and "project" concepts, only minor refactoring was needed.

5.3 Scenario module

The goal of the scenario module is to manage a scenario, whether the scenario shall be e.g. MSDL (SISO, 2008), Half-Life or Virtual Battlespace.

There are a set of requirements that each module must be able to support;

- Add components.
- Remove components.
- Search for components.
- Import a scenario.

- Export a scenario.
- Path selection for scenario image and scenario path.

Due to the structure of the requirement set, each scenario module may store and manage its components in an arbitrary format. Therefore, the back end can keep track of the abstract concept of a “scenario module”, without having to be programmatically locked to e.g. MSDL or Half-Life.

There are three scenario modules present in MSDLSE; MSDL, OGC-KML and Virtual Battlespace 2 (from here on referenced as VBS2).

5.3.1 MSDL scenario module

The MSDL module written stores its data in the same fashion as the MSDL specification is written; as a tree, where the root node is "MilitaryScenario", per the MSDL specification.
Nodes and their attributes in the specification are likewise represented in our internal tree database as classes, utilizing the API ITree (Butler, 2007). The ITree API allows storage of arbitrary objects, making it a prime component for general scenario module storage. See Appendix A and B for an overview of the structure.

There are several advantages to this approach;

- Simpler to traverse the scenario components
- Simpler to transform to XML
- Logical structure
- Simpler to add more components
- Simpler to debug the code due to its structure

The disadvantages to the design are;

- Time consuming to insert components
- Time consuming to search the tree
- Higher complexity for importing from XML.

The estimated time to perform inserts and searches are estimated to O(n) (Helmbold, 2003).

The initial design of MSDLSE utilized pointers to particular components that were used often, e.g. the MSDL tree’s “Units” node. This approach allowed an O(1) insert time and O(n) for searches. The functionality was later removed due to the code complexity of the design approach.

The availability of the XSD-files (SISO, 2008) for the specification allows MSDLSE to (try to) validate against the XSD-files, if present.
Per project management are the XSD-files replaceable. The advantage to this flexibility is updating possibilities for newer versions of the XSD-files and the MSDL specification.
The validation occurs before imports and after export. MSDLSE will not continue to import a scenario if its validation has failed.

5.3.2 OGC-KML scenario module

OGC-KML (Open Geospatial Consoritum, 2008) is a markup language, structured according to XML. It is developed by Google, and was approved as an industry standard by the Open Geospatial Consortium (OGC) on April 14th, 2008.
OGC-KML is used to express geographical items and placeholders in web applications and in Google Earth.
Due to legal reasons, the exposed OGC-KML support in MSDLSE is localized to exporting arbitrary MSDL scenarios in OGC-KML form.

5.3.3 Virtual Battlespace 2 scenario module

VBS2 scripting is a proprietary scripting language (Lundgren, Perak, Michalski, Ullner, 2007) is developed by Bohemia Interactive for Virtual Battlespace 2. The VBS2 script files are the only available API or SDK for VBS2 (Olsson, Michalski, 2008).
Due to time restraints, the exposed VBS2 script file support in MSDLSE is localized to exporting arbitrary MSDL scenarios in VBS2 script file form.

6 Implementation of the MSDL Scenario Editor

This chapter describes the implementation details of MSDLSE. The design decisions done in the previous chapter are used as the basis of the implementation.
Before reading the implementation details, it is important to know what functionality MSDLSE has; this information will be noted in section 6.1.
The application presented below is implemented with Microsoft Visual Studio in the language C# (Hewlett-Packard, Intel, Microsoft, 2006).

6.1 MSDL Scenario Editor functionality

MSDLSE support a wide range of functionality, as listed in Table 6A. See the rest of this section for description and the rest of the chapter for implementation details.

Functionality list	
	FUNCTIONALITY
Section 6.1.1	Project creation
Section 6.1.2.1	Scenario creation form vector files
Section 6.1.2.2	Scenario creation from bitmap files
Section 6.1.2.3	Scenario creation from WMS server
Section 6.1.3	Import project Import scenario
Section 6.1.4	Export as MSDL Export as OGC-KML Export as image Print image
Section 6.1.5	Validate current senario Validate external scenario
Section 6.1.6	Map preview Tabs MIL-STD-2525B Dynamic coordinate information Dynamic resizing of maps MSDL component editing
Section 6.1.7	Logs
Section 6.1.8.1	Application properties
Section 6.1.8.2	Project properties
Section 6.1.9	Keyboard shortcuts Readme file

Table 6A – List of functionality and respective section for implementation

6.1.1 Project creation
MSDLSE enables project creation and management. A project is composed of its project settings and a set of scenarios. A project is required for scenario interaction.

6.1.2 Scenario creation
MSDLSE allows creating a scenario in three different ways;

- From vector graphic files.
- From a bitmap file.
- Through a Web Map Service (WMS) (Open Geospatial Consortium, Web Map Service) server.

All three ways to create scenarios allows the user to see a preview picture of the selected image before confirmation.

6.1.2.1 Vector graphic files
The current version of MSDLSE allows only creating vector images from Shapemap (.shp) files.
The vector technology enables the possibility to zoom in and zoom out from the image. This feature is only available when a scenario is created from vector graphic file or WMS server.

6.1.2.2 Bitmap file
Creating scenario from a bitmap file allows the user to create a scenario from e.g. a stored bitmap, JPEG, GIF or TIFF file. The user can as well specify an URL address to download an image from the Internet.

6.1.2.3 WMS server
The WMS server produces and returns an image based on a query that is sent to the server.

6.1.3 Importing
MSDLSE is able to import projects and MSDL scenario files. The XSD-files are required to be able to import a MSDL file. If the imported project has one or more scenarios saved, they all will be displayed loaded in MSDLSE.

6.1.4 Export
There are four ways to export a scenario;

- MSDL file.

- OGC-KML file.
- VBS2 file.
- Image file.

All MSDL files are fully compatibly with the MSDL specification and therefore are possible to import in other MSDL editors.

The OGC-KML file allows seeing the scenario through web based applications or in e.g. Google Earth.

The VBS2 file allows using the scenario through VBS2 (Bohemia Interactive, Virtual Battlespace).

An image export is a PNG (World Wide Web Consortium, 2003) based picture that can be imported in any picture editing program. As a complement to export images, it is possible to print the image to paper.

6.1.5 Validation

MSDLSE provides a validation function for MSDL scenario files. It is possible to validate an external MSDL file at any time while running MSDLSE and possible to validate a scenario that is open for the moment. All scenarios that MSDLSE saves in MSDL form will be validated against the XSD-files (SISO, 2008). A scenario will be saved, regardless if it validates. However, there is a risk that MSDLSE will not be able to import that scenario in the future.

To be able to use the validate functionality, there has to be XSD-files in the "xsd" folder for a project, in the application start-up folder, or wherever the XSD-file setting is set to. This solution allows easy replacement of the XSD-files to other versions.

6.1.6 Interface functionality

The current version of MSDLSE (V1.0) is using GDC coordinates.
By default, a scenario's area is the "entire world". That is because the application can't decide where in the world the map exists. If a user recalibrates the GDC coordinates, MSDLSE will automatically recalibrate all objects that are in the scenario.

To be able to work with more then one scenario at the time, MSDLSE provides tabs that allow the user to easily switch between scenarios in the project.

MSDLSE support appendix A of MIL-STD-2525B symbols.
In a tree view, the user can select the desired symbol identifier and see a preview image of the symbol.
There is possibility to directly enter a symbol identifier if the symbol identifier is known.
The MIL-STD-2525B symbols are easily replaceable with other versions of MIL-STD-2525B symbols and even with other symbol standards.

GDC coordinates are displayed in the bottom of the application, of where in the world the cursor is, which is useful if a user desire to place an object more precise on the map.

It is possible to change all attributes of a unit object, add or edit a force that the unit belong and add equipment based on a MIL-STD-2525B symbol identifier. All attributes that can be edited have default values and descriptions that are taken from the MSDL specification.

All objects that are on the map can be selected for property editing, movement around the map and copying to other scenarios.

6.1.7 Logs

Every errors and important messages that occurs in MSDLSE will be logged into two log files.

- error.txt – logs all error that can occur in MSDLSE.
- message.txt - logs all other relevant information.

The files are located in every project path and in the MSDLSE folder.

6.1.8 Properties

There are two types of property settings in MSDLSE;

- Application properties.
- Project properties.

6.1.8.1 Application properties

These property settings points to where the following files are located;

- XSD-files.
- MIL-STD-2525B symbols.
- Log files.

6.1.8.2 Project properties

The project property settings points to where the project storage is located.

6.1.9 Keyboard shortcuts

To allow faster interaction with MSDLSE, shortcuts are provided for most common features to;

- Save a scenario.
- Cut out a MSDL component from the scenario.
- Paste out a MSDL component from the scenario, after it has been cut out.
- Delete a MSDL component from a scenario.

There is as well “readme” file in the executable directory where all shortcuts are explained more carefully.

6.2 User interface

This section describes the implemented user interface, both programmatically and visually how the user shall perceive the interface.

6.2.1 Implementation of interaction with the user

As noted previously in chapter 5, the user interface is implemented graphically and MSDLSE’s features are summarized in the previous section. The following section describes the implementation of interaction with the user. See figure 6A for a screen shot of MSDLSE and further down for explanation of the numbers.

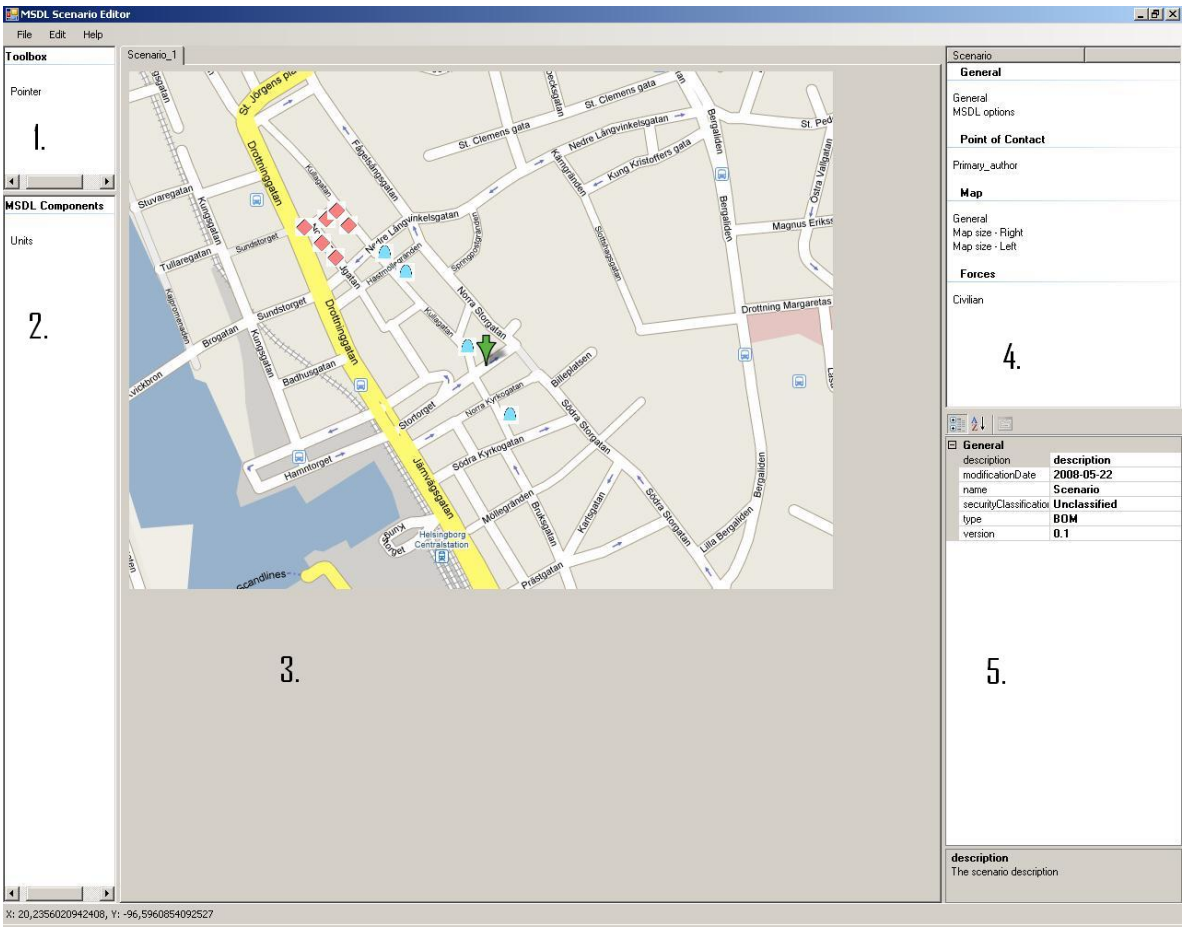

Figure 6A – Screen shot of MSDLSE

When MSDLSE is started, the only visible and interactive object is the menu. The menu allows;

- Ability to change the application's properties.
- Help and copyright information.
- Validation of external MSDL files.
- Project creation.
- Project importing.

The application's properties are a selection window for different settings, as described in section 6.1.8.1. The user enters paths to said files or folders or by using a browse button.

The help information is a briefly detailed "readme" text file, containing information about scenario creation, scenario export and more.

The copyright information is information from the "About" component, briefly mentioned in section 5.1.1, detailing the version of MSDLSE, its copyright and STS ownership.

When attempting to validate an external MSDL file, MSDLSE will utilize and require the accompanying XSD-files (SISO, 2008).

A project in MSDLSE is initiated by either creating a new storage place or by importing a previously created project.
When creating a new project, one selects the storage place for the project. A MSDLSE project file will be created in this directory, storing the storage path and (later) storing scenario image and scenario file paths.

When a project has been created or imported, the menu will allow more selection items;

- Project properties
- Saving the project.
- Closing the project.
- Scenario creation.
- Scenario importing.

Of course, the previous menu items are as well selectable.

The project properties allow the user to specify a new project storage path. The project storage path is the path of where MSDLSE will look, when trying to use certain files (detailed below).

The project can be saved and closed, allowing users to do their work, save, close and then continue working with another project.

When a project has been successfully set up or loaded, the user shall begin using scenarios. This can either be achieved by importing an existing MSDL scenario file or by creating a new scenario.

MSDLSE will, as previously noted, on import validate the MSDL scenario file before any importation is done. This allows MSDLSE to verify whether the MSDL file is a genuine MSDL file and contains proper information for MSDLSE to accurately generate a scenario.

When creating a scenario, one is presented with a map source selection window, see figure 6B. The window allows multiple sources of map information; vector files, normal image files and from a WMS (Open Geospatial Consortium, Web Map Service) server.

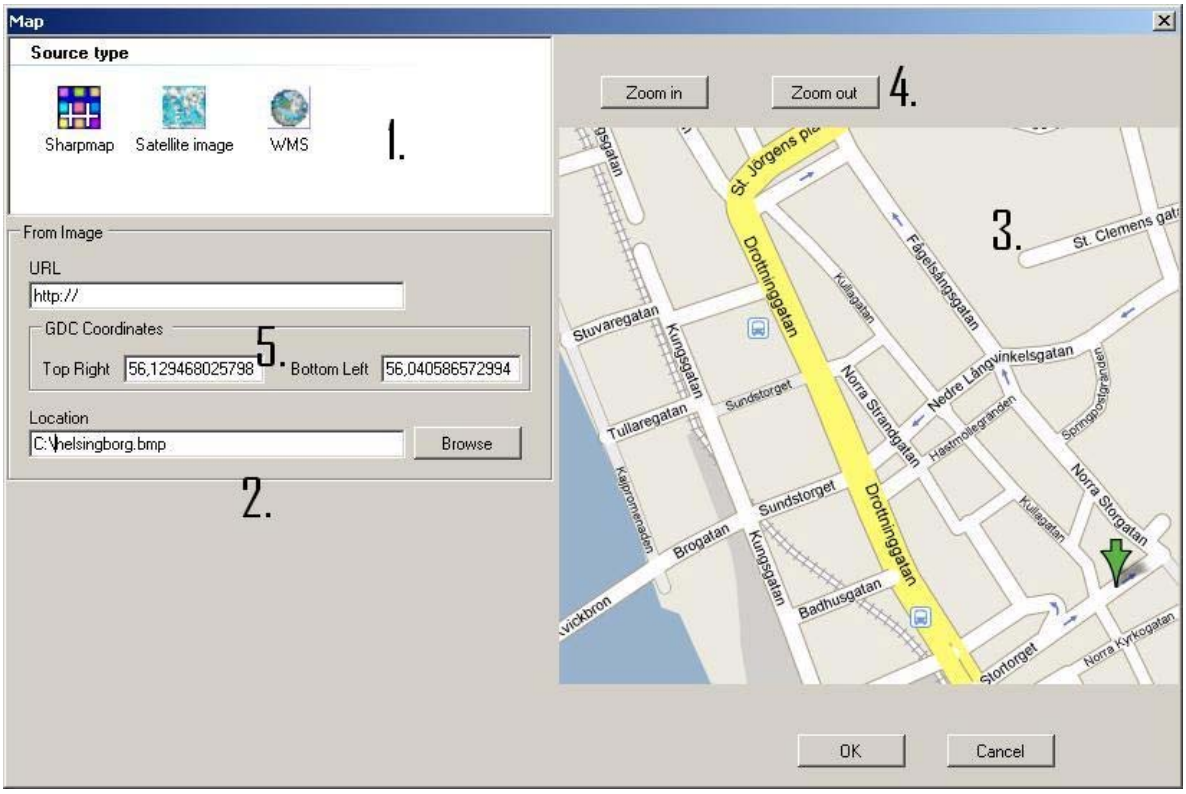

Figure 6B – Screen shot from MSDLSE when creating a scenario

The user selects the desired source type (figure 6B.1) and further the desired scenario image (figure 6B.2). A preview of the selected scenario image will be displayed (figure 6B.3), allowing the user to make up his or hers mind before basing their scenario on said image. When using vector files or a WMS server, MSDLSE will allow the use of a zooming function (figure 6B.4) of the scenario image in the preview. This allows the user to load a vector file representing the world and zooming in to e.g. Paris or London, without creating a pixilated image.
As well, the user is able to input GDC (figure 6B.5) coordinates of the preview image area, or else it will default to the representation of the "entire world". Vector files and the use of a WMS server allow automatic updating of the GDC coordinates, without the need for user interaction.

When a scenario image has been selected by the user, MSDLSE will allow more interaction;

- The menu will be expanded to include;
  - Saving the scenario file to the project
  - Exportation of the MSDL scenario.
  - Exportation of the image file.
  - Print of the image file.
  - Validation of the (current) MSDL scenario.
  - Closing the scenario (and possible removing it from the project).
- Possibility to change the properties for the scenario

- Possibility to add MSDL components
- Possibility to change the properties of individual MSDL components.

Of course, the user has the choice to add additional scenarios. Each scenario will be displayed in a tab view, allowing multiple scenarios per project.

The user has the choice to save a scenario, without having to save the other scenario files in a project. As well, the user may select to export the scenario MSDL file, for another MSDL scenario rendering application or for later. MSDLSE will, on saving and exporting, automatically validate the output file. As well, the user has the option of validating the currently constructed scenario. Of course, MSDLSE will try to keep the tree valid; validation is just a precautionary method.

The user has the ability to export the scenario image file and as well print the picture – directly from within MSDLSE.

The user is represented with two choices when closing a scenario; either by closing it from the tab view or close it and remove it completely from the project.

When the scenario has been created, MSDLSE will create default values for MSDL nodes, e.g. by stating that the MSDL scenario is “Unclassified”. However, as the user might want to change the value of “Unclassified” to “Classified”, MSDLSE will present the user with a property view for MSDL nodes and another property view for the MSDL nodes’ attributes. MSDLSE and the C# runtime shall ensure that the inputted values are correct – from a MSDL specification point of view.

When the user is satisfied with the scenario’s properties, the user may opt to add MSDL components. The user selects the type of MSDL component – the only implemented is “Units”.

The user is greeted with a window, see figure 6C, where he or she selects the Unit’s type of force – unknown, neutral, friendly or hostile.

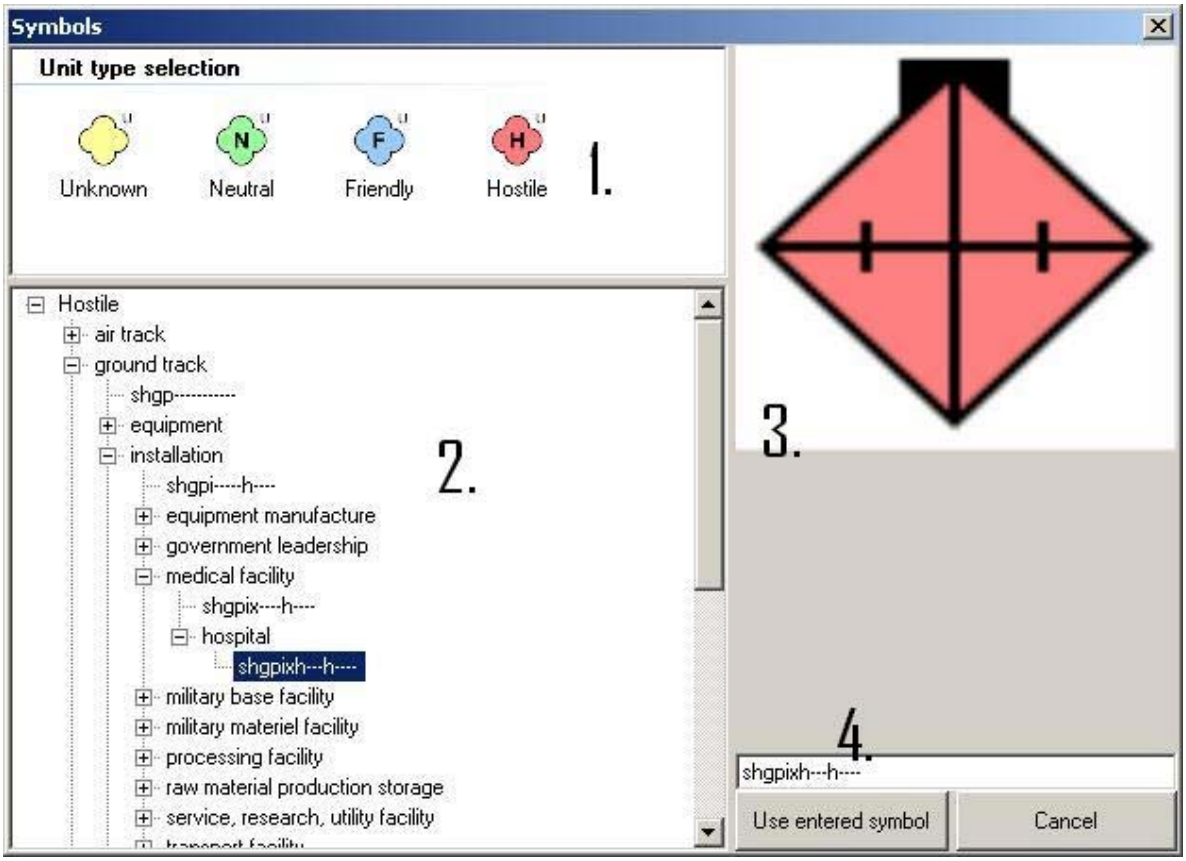

Figure 6C – Screen shot from MSDLSE when selecting MIL-STD-2525B symbol identifier

When selecting a force (figure 6C.1), MSDLSE will create a tree (figure 6C.2) of where all unit MSDL components are displayed that have the corresponding force. The tree will display MIL-STD-2525B symbol identifiers, allowing the user to select which unit that shall be added. MSDLSE will, when the user selects desired MIL-STD-2525B symbol identifier, display the image of the symbol identifier in a preview window (figure 6C.3). Additionally, the user may input the symbol identifier (figure 6C.4) and MSDLSE will automatically display the correct symbol identifier in the preview window.

When symbol identifier is selected, it is possible to place the unit on the scenario. The icon of the mouse cursor shall as well be changed, drawing attention to where and what shall be placed.
The mouse cursor's icon will remain as such until the user selects the Pointer tool in a tool box.

When the pointer tool is selected, the user can select added MSDL components. When selecting a MSDL component on the scenario, MSDLSE will, as previously with the scenario, display the MSDL component's properties. The user may as well e.g. move the MSDL component within a

scenario by holding down a key on the keyboard and dragging the MSDL component.

6.2.2 Programmatic implementation

The user interface is, as described in section 5.1.1, broken down by functionality.

6.2.2.1 Main window

The main window is divided in five sections;

- The tool box (figure 6A.1).
- The MSDL component selection (figure 6A.2).
- Tab and image view (figure 6A.3).
- Selection view for nodes (figure 6A.4).
- Property view for said nodes' attributes (figure 6A.5).

Additionally, the main window contains a status bar, presenting general information about the scenario, and a menu, as described above.

When a scenario's properties for the size of the map changes, MSDLSE will automatically recalibrate MSDL components, which have a coordinate, according to the new map size. That is, the MSDL components in a scenario shall still belong to the scenario, by having their coordinates in the scenario, and as such shall be updated.

When creating windows, tabs and more, MSDLSE will create listeners for key presses and mouse button clicks. These listeners enable the application to listen for actions such as "the user has clicked on the map" or "the user has selected this value", and act accordingly.

The tool box and MSDL component selection are hard coded values in MSDLSE. As such, they are not dynamically generated according some specification. The reason for this decision is that there would have taken too much time, for little benefit, to dynamically generate the items. As there are e.g. no other MSDL components added to MSDLSE (in regards to Units), it was not vital for anything dynamic or highly flexible.

6.2.2.2 Input box window

The input box window is a simple input box; requests a type of information from the user, which they in turn provide (hopefully, at least). The input box will store previously entered values (if it's in the same category as a previous

input), and write those values to a file on the hard drive. Of course, the input box will load previously entered values from said file.

6.2.2.3 Scenario creation image window

The basic purpose of the window is to guide the user and force them to create a scenario based on its origin, similar to a wizard.
As noted previously, MSDLSE allow creating a scenario from vector files, bitmap file and from a Web Map Service (WMS) server.

The bitmap file can be any random image file, and MSDLSE allow downloading the file from the Internet or by browsing a hard drive.

The vector files are mathematical geometrical data to represent images. As such, there are no "pixels" in a vector file, allowing zoom functionality without compromising the image. MSDLSE utilize the open source Sharpmap (Nielsen, 2008) component for vector file rendering.

The WMS server functionality is basically the same as the vector files and is rendered in the same fashion. The WMS server is just a remote location for the vector file storage, allowing multiple sites to use only one storage facility.

Note that since multiple vector files can be selected for one scenario, it is not possible to automatically create an unambiguous name for the scenario file. As such, MSDLSE will create a file called "Image123456.jpg", where "123456" is a randomly generated string.

6.2.2.4 MIL-STD-2525B selection window

The window's purpose is similar to that of the scenario creation image window. The intent is to force the user to make an initial selection, and then be presented with another selection. This lowers the risk of having the user be overwhelmed with information.

The basic functionality is that it calculates the correct symbol identifier and presents it for the user.

6.3 Back end

The back end is mainly a communication flow transferee. However, it manages several other important parts of MSDLSE, such as settings and logs.

6.3.1 Project storage management

As previously noted, the project storage is essentially the concept of a "database". The database store scenario modules (or handlers).

The database gets notified when the scenario changes (in the tab view), allowing fewer information to be sent whenever there is a query from the user interface.

The database forwards information from the GUI to respective handler and sends back information coming from said handler, if the information is intended for the GUI.

The database also keeps track of the project's settings and the application's settings.

6.3.2 Settings

The Settings class of MSDLSE allows arbitrary storage of settings. Primarily, the project has an instance of the class, where all project specific information is stored, such as the location of the log files or XSD-files. Secondarily, it is also able to store the application's settings. This enables MSDLSE to store all relevant information and files for all projects that utilize MSDLSE. That is, if the project instance wishes to see where the XSD-files are, but cannot because the setting isn't set (for whatever reason), MSDLSE will default its XSD-file path to the path that the application's settings are set to. In essence, the mechanism is intended as a fail safe.

6.3.3 Logs

There are two types of log files in MSDLSE; error.txt and messages.txt. The former file will receive information whenever an error (in the magnitude of exceptions, missing crucial files etc). The latter file will receive messages from MSDLSE, but only if it's for informational purposes.
That is, if a node in a MSDL file validates unsuccessfully due to a faulty value, MSDLSE does not treat it as an "error" but rather "something went wrong, but it's only worth informing about".

The log files are accessible from the entire application, allowing each component to inform the user, regardless of where said component is put.

6.3.4 General functionality

As the code for applications (in general) can grow to high numbers, it is useful to keep a repository of general and commonly used functionality. Indeed, MSDLSE uses the same paradigm.
The general functionality class allows other components to;

- Verify an arbitrary string or number according to some boundaries.
- Calculate distance differences between two points

- Resizing of an arbitrary bitmap
- Deserialization of XML files.
- Search an ITree (Butler, 2005) after a specific type (and as well remove).

Similarly as the logging facility, the general functionality allows cross-component access.

6.4 Scenario module

As described in section 5.3, each scenario module is required to implement a set of requirements. This is enforced by using the abstract concept (interface in C#) of a scenario module "handler", where the back end stores and references the modules as such.

A handler enables;

- Adding components.
  - Simple adding of an object.
  - Adding an object, where another object's information is required as well.
  - Through an ITree.
- Removing components.
  - Removing by object
- Searching for components.
  - For an identical object.
  - For any object containing some specific information (retrieving one or multiple objects).
  - For any object at a specific coordinate.
  - For a specific type of object (retrieving one or multiple objects).
- Importing a scenario.
  - By object (that is, from file or source from a network etc).
- Exporting a scenario.
  - By object (that is, from file or source from a network etc).
- Path selection for scenario image and scenario path.
  - The absolute path to the scenario image file.
  - The absolute path to the scenario file.

6.4.1 MSDL

As described earlier, the MSDL support range from generation, to importation to exportation.
The implemented nodes can be viewed in Table 6B. Note that all nodes are not implemented; firstly and foremost are required nodes implemented, per the specification. Secondly, the implemented (optional) nodes are added, per the

MSDL elicitation process described in chapter 2. Note that not all node attributes are added, unless required.

AreaOfInterest	CommunicationNetInstance	CommunicationNetInstances
Coordinate	CoordinateData	Disposition
Environment	Equipment	EquipmentItem
ForceRelation	ForceRelationData	ForceSide
ForceSides	FormationData	FormationPosition
GDC	MilitaryScenario	Options
OrganicRelation	OrganicRelationData	Organizations
Owner	OwnerData	OwnFormation
Poc	Relations	ScenarioID
Status	SupportRelation	SupportRelations
Unit	UnitPosition	Units

Table 6B – Implemented nodes from the MSDL specification.

The MSDL module is, as described in the design chapter, implemented by utilizing the ITree (Butler, 2005) API (from here on referenced as “the tree”). Each implemented node, per table 6B, is described in MSDLSE as a class. An instance of said classes will be created and inserted in a stored tree. The stored tree is stored internally in an instance of the MSDL handler.

The scenario module also stores settings for the current project. This allows the module to keep track of e.g. the path to the MIL-STD-2525B directory, allowing MSDLSE to set an image for an object, based on its MIL-STD-2525B symbol identifier.

Each item that shall be display in the property view, described earlier in this chapter, will have a description and its corresponding category, set in each variable instance. Each object’s attributes are set to its most appropriate value depending on the recommended value in the specification and our consideration of possible values.

As explained in section 6.1.4, it is possible to select which MSDL components that shall be exported. As such, the tree must be purged from any object that has been selected for removal. As well, any object references to said removed object must as well be removed. This scheme is quite an expensive action but necessary per the MSDL file that shall be exported. No node is removed from the original tree, allowing users to further interact with the scenario and its components.

The MSDL export will, beyond cleaning the tree;

- Add the appropriate xmlns (see section 4.3) tags for the XML.
- Traverse the tree and for each node add it as a XML node.
- Output the XML file.
- Validate the XML file.

MSDLSE will on import;

- Validate the XML file.
- If the file validates successfully, continue importing. Otherwise, abort.
- MSDLSE will establish the same tree and create the same structure as it would have if the scenario was created from scratch in MSDLSE.

Note that an imported scenario will cause MSDL components to be displayed incorrectly, insofar that they adhere to the specification's requirement that certain characters from the MIL-STD-2525B symbol identifier shall be omitted. The scenario module will *not* attempt to calculate a unit's force by e.g. "hostile" or "friend", as such information is not supported by the MSDL specification.

The validation is done by utilizing the XSD-files (SISO, 2008) accompanying the specification. The validation is done by adding the files "MilitaryScenario.xsd" and "ModelID_v2006.xsd" to C#'s XML validation component, which notifies if and what error has occurred. However, the total 8 XSD-files must be present for the validation to be successful.

The implemented coordinate system is Geodetic coordinates (GDC) (Anon, Geodetic and geocentric coordinates). The elevation attribute in GDC has no bearing on the visualization and generation of the map or units.
Note that the scenario module will use -179 in GDC, regardless of its inaccuracy, as previously noted.

The MSDL specification requires the entire MIL-STD-2525B specification for compliance. However, due to time restraints, the supported section of the MIL-STD-2525B specification is Appendix A.

Certain elements of the MSDL specification require the use of unique identifiers. These identifiers are generated by the C# API System.Guid.NewGuid (Anon, System.Guid.NewGuid). As such, if a collision is discovered in said API, MSDLSE will also be affected.

MSDLSE will perform value checks on information gathered from the user. A value change is allowed if said value abides by the specification. Any other sanity check lies on the user. E.g., a vehicle's speed is restricted to 0-999999, where the unit system is depending on the vehicle disposition (km/h for ground elements, nmi/h for air etc). MSDLSE will enforce this check, so the user shall not be able to enter -1 (negative one). However, MSDLSE will not (and shall not) care whether the user has entered 999998 for (the representation of) a normal human on the ground.

A minor arbitrary restriction in the scenario module is that "point of contact's" email is restricted to 10 (ten).

6.4.2 OGC-KML

The OGC-KML (Open Geospatial Consortium, 2008) support in MSDLSE is localized to exporting arbitrary MSDL scenarios in OGC-KML form, with hard coded information.

Exported OGC-KML files describe;

- The symbol for a unit.
- The unit's name.
- Description of the unit.
- The unit's coordinate.

The above listed information consists of (in the OGC-KML file) style nodes for symbols and placemark nodes for units.

The style node is structured as containing;

- Unique identifier for symbol (MIL-STD-2525B symbol identifier).
- The symbol's path, for visualization in e.g. Google Earth.

The placemark node is structured as containing;

- The unit's name.
- The unit's description.
- A reference to the unique identifier for symbol (in a style node).
- The unit's coordinate in GDC (Anon, Geodetic and geocentric coordinates) coordinates.

There is one style node for each unit with unique symbol identifier. So e.g., if there are three units, where two represent a “helicopter” and one represent a “ground unit”, there will be two style nodes. Respectively, there are three placemark nodes, considering the three units have unique properties, such as coordinates and description.

Note that the OGC-KML file is *not* validated against any XSD-files.

See Appendix D for an example of an OGC-KML file.

6.4.3 Virtual Battlespace 2

The VBS2 script file support in MSDLSE is localized to exporting arbitrary MSDL scenarios in VBS2 script form, with hard coded information.

Exported VBS2 script files describe;

- Existing forces in the scenario.
- A unit with VBS2 unit identifier.
- The force the unit belong to.
- The unit’s coordinate.

Note that the VBS2 script file is *not* validated against any validation files.

See Appendix F for an example of a VBS2 script file.

7 Results

This project was originally planned for 20 weeks, with 3 milestones, with the deadline 30th of May. The project started week 6 in 2008 and ended week 25, 2008. For more information about the project and work methods see chapter 1 and chapter 2.

7.1 Goals

The goals in this thesis are divided in three groups;

- Acceptance requirements.
- Saab Training Systems goals.
- Our project, thesis and course goals.

The goals presented in this section were considered as challenges, due to the lack of knowledge beforehand. The goals were challenging, fun and interesting to complete. Some goals were in themselves hard and some were easy, but most of them were challengingly interesting and we experienced that most of the challenges were easy and fun. Hard goals were e.g. the OGC-KML export and vector graphic support, but were later simplified by its interestingness.

7.1.1 Acceptance requirement

The acceptance requirements are the requirements that have to be fulfilling to pass this project.
The acceptance requirement that was delivered to us from STS was following;

- Create a common reference model for serious gaming and a common reference model for real-time strategy games.
- Create a proof-of-concept application that can generate scenarios in MSDL format.
- Export of scenarios in MSDL form.

These acceptance requirements have been confirmed by STS to be fully fulfilled. For more information about the acceptance requirements and goals see chapter 1.3.2.

7.1.2 Saab Training Systems goals

STS long term goal is to increase the relation between themselves and Lund University, LTH.
STS' goals with this project are;

- Simplify the preparations for training sessions, by using the MSDL standard. The system should be able to generate, import and export

training scenarios on MSDL format and then be able to load that scenario file in all games and systems connected.
- A Strategy Serious Gaming Common Reference Model (SSGCRM) shall be created for real-time strategy based games, such as World in Conflict or Command & Conquer.
- A Serious Gaming CRM (SGCRM) shall be created for all games, together with the cooperation thesis (Olsson, Michalski, 2008). The SGCRM shall describe the lowest common denominators for games in respective genre.

All of these goals have been confirmed by STS to be fulfilled. For more information about STS' goals, see chapter 1.3.1.

7.1.3 Our project, thesis and course goals

Our goals for this project, thesis and course were;

- Minimally fulfilling the acceptance requirements in section 1.3.2.
- Use all previous knowledge and incorporate it into this thesis paper
- Additional functionality to the application in the form of satellite image rendering; standardized symbology; extended import and export functionality; a simple code base for further development.
- Creating a run-down on the information gathered and lessons learned while implementing the MSDL specification.
- Deliver the product and this paper in estimated time.
- Become co-writers in a paper for the MSDL standard, which will be presented at a SISO Simulation Interoperability Workshop in August of 2008.

All of the goals have been confirmed by STS to be completed, except of the last goal: "Become co-writers in a paper for the MSDL standard, which will be presented at a SISO Simulation Interoperability Workshop in August of 2008", due to the obvious time difference.

7.2 Reference models

A reference model is an abstract model of information that can be exchanged between several systems. In this thesis we have developed two reference models.
The first model is a real-time strategy common reference model that is suitable for tactical games and the second model is a common reference model which is developed together with cooperation thesis.

7.2.1 Strategy Serious Gaming Common Reference Model

Our real-time strategy serious gaming common reference model was originally developed to fit teal-time strategy games. As mentioned in chapter 3, our real-

time strategy reference model contains larger entities that is composed of multiple child entities and attributes that the children inherit. See Table 3A – Overview of real-time strategy gaming common reference model.
The Root node contains 5 Childs; “Objects”, “Communications”, “Weather”, “Map” and “statistics” All this children have their own children and attributes. For more detailed description of all children see section 3.1.

A relationship model was also developed for the reference model. This model contains relations between all children, what direction the information flows and actions they can perform to each other. All relations and cardinality can be seen in chapter 3.2.

7.2.2 Serious Gaming Common Reference Model

As mentioned in section 3.3, the common reference model was developed together with the cooperation thesis. This reference model is merged from our reference model that is based on real-time strategy games and our cooperation thesis’s reference model that is based on first-person shooter games.
The advantage with a common reference model that fits most common games is that there is no need to create a reference model for every game.

The relationship model for the common reference model contains relations between all children. This relationship model is similar to the real-time strategy relationship model except from three blocks that contains child nodes. Section 3.4 contains more information about the general relationship model.

7.3 MSDL

The specification requires and uses several standards and concepts, which can be seen in table 4A. To support the entire specification, all standards have to be included. Due to time limitations we chose to implement parts of the specification. What parts we chose and why can be seen in chapter 4.1.

The specification allows much flexibility. E.g. the specification allows choosing between four types of coordinate systems. While flexibility is always good in advanced MSDL rendering software, the flexibility may cause issues for visual representation, as the information (mostly) contradicts each other. E.g. a unit doesn’t necessary need a coordinate, and therefore will be difficult to represent visually in a scenario.
This is not necessary an issue, this kind of flexibility allows the programmer to decide how the application shall work, but can cause incapability with other MSDL applications.

The specification is easy to read, use and follow. It is well-written and unambiguous most of the time. However, it should be noted that we believe

the specification is written for and by people deeply involved in military training, simulation organizations and the military's usage of terms and concepts. Therefore, it may be difficult for a layman to understand and use the specification to its full extent.

7.4 MSDL Scenario Editor

The MSDL Scenario Editor is a proof-of-concept application and therefore, together with STS, we decided that the focus with this application will be quantity in functionality rather then quality. That means that the MSDLSE have many features, though might not be 100% bug free.
The GUI is clean and free from unnecessary information; the property window has information about every attribute that can be changed and a preview window for MIL-STD-2525B symbols.

MSDLSE is split up into three parts;

- GUI.
- Back end.
- Scenario module.

Scenario module can be replaced as desired. That allows the programmer to change to another scenario module, such as VBS2.

The entire application is designed with expansion in mind. That means;

- Easy to replace XSD-files for verification, in case a new MSDL version will be developed.
- All Scenario modules can be added or replaced from the back end.
- All MIL-STD-2525B symbols can be replaced without editing the source code.
- New export functions are possible to add such as OGC-KML or VBS2 script files.

Although this thesis is completed, it is possible to further expand MSDLSE and MSDL (see chapter 8 for MSDLSE and MSDL suggestions).

8 Conclusion

This chapter expresses our personal thoughts and reflection of this thesis, MSDL, MSDLSE and the cooperation thesis (Olsson, Michalski, 2008).

8.1 Thesis reflection

8.1.1 Work methods

As described in earlier chapters was this project broken down in four phases. The initial "information gathering" phase was conducted swiftly, a week before the time deadline for the phase. The "production" phase was as such started earlier, and spanned the rest of the project time. The "presentation" phase started on time and ended as predicted. The "documentation" phase began a few weeks after it was initially planned, but was successfully delivered on time.

Moving the "production" phase a week earlier and spanning the rest of the project made the entire phase much more appealing. The time shift allowed us to experiment with different solutions for the application. The large time frame allowed the "documentation" phase to be postponed with little consequences, especially with the mandatory "documentation writing" day, as explained previously.

This thesis and the cooperation thesis received one computer each at STS. This had certain advantages and as well disadvantages. The advantages are that we conducted all research and development together, enabling us to gain the same knowledge simultaneously. The main disadvantage is that only one person can "use" the computer, when separate computers may have been more time effective. When needed, personal laptops were brought and used (though not for MSDLSE development).

8.1.2 Communication

The communication in the project went fine with almost all parties. The communication with the cooperation thesis, staff at STS, the thesis examiner and the technical supervisors went fine.

This thesis was conducted at STS' facilities, enabling us to easily communicate verbally (as well as electronically when needed) with the staff at STS, the technical supervisors and the cooperation thesis. Consistent updates and correspondence between us and the thesis examiner where held with e-mail, with adequately short response time.

Comments from those involved in MSDL have been few, with the exception of Per Gustavsson. Per visited STS at the middle stage of the project and gave suggestions, commentary on MSDLSE and responded to the MSDL comments we had at the time. Per also submitted the thesis' abstract to the SIW conference. Shortly after Per's visit, we posted a forum post on SISO's website with MSDL commentary, with the purpose to get more knowledge about MSDL for MSDLSE and notify SISO members what the thesis was about. Unfortunately, we received no replies.

8.1.3 Scope

The thesis's scope was very limited, at first. We focused in the initial stage of the project to finish the acceptance requirements and only them. As the project progressed, STS gave more suggestions for MSDLSE's capabilities and this project's tasks. Due to obvious time restraints in the thesis, we discussed relevant features and tasks with STS, and changed the scope accordingly.

8.1.4 Risk analysis

A major risk in this thesis was the low amount of knowledge we had about the C# language, MSDL and reference modeling. Fortunately, we had experiences in Java, which have similar syntax and functionality, allowing us to quickly familiarize with C#. While we had never used MSDL or anything similar, the structure of the document allowed a quick learning curve. The reference models were simple to develop, due to the close relationship with the cooperation thesis and the technical supervisors.

Another major risk in this thesis was possible absentees, possibly forcing time schedule delays. However, the constant communication and division of work enabled us to complete tasks in time.

Finally, this thesis has been instructive and interesting to write and do.

8.2 MSDL Suggestions

The MSDL specification is fairly new and there is always space for improvements. In this section we will describe suggestions that we think can be useful for the next MSDL version. We believe that our suggestions would make the MSDL specification more complete and useful for our application or already existing applications.

- In the MSDL specification, under the "OwnFormation" node, there is a "FormationType" node that describes what formation a unit has. However, there is another node under the "OwnFormation" node that is named "FormationData". This node describes exactly that information "FormationType" node does (and even more). Therefore we believe that

the "FormationType" node should be removed from the MSDL specification to optimize the code.

- C-BML support – as mention in section 4.2 the C-BML is a Battle Management Language. The C-BML allows mission specific information to be sent. Assuredly this feature will be implemented in next version of MSDL specification, but we miss this function in our thesis.
- For more advanced scenario editing, we would like to add a relative map surface. Our suggestion would be to make a list of coordinates in the MSDL specification that would represent the edge of the map.
- Triggers – triggers is a certain area that perform something if a certain condition is met. This feature is very handy if there has to be e.g. a mine field or an air-strike. Instead of placing hundreds of mines or bombs one-by-one, it can be done by defining an area.
- Event – an event is something that happens at a given place and time.
- In the current MSDL specification, the only association a vehicle has with a unit is by a reference. That doesn't necessary need to mean that a unit is in the vehicle. E.g., a unit can own a vehicle but not necessary be in the vehicle. We would like to make this kind of feature more legible in the next MSDL specification.
- Time changes are not an important feature, but since the MSDL specification contains a lot of weather information it would be a good complement. E.g., at the start of the scenario the weather is rainy and one hour after the scenario is started the weather is sunny. This feature can be used for more purposes then just weather changing.
- Maybe the most important feature (beside C-BML support) that we consider is the time aspect. This feature can be combined with almost all things that are created in MSDL and would be very valuable if it would be implemented. E.g., a unit can't make things happen until a certain time. This feature would fit perfectly with the C-BML support.

These features are just our own suggestions of how the MSDL specification could be improved. Beside the C-BML support (Anon, 2008, Future MSDL Focus), there is no information if someone of these features will be added to the MSDL specification.

8.3 Possible ideas for the MSDL Scenario Editor

While MSDLSE was developed as a proof-of-concept application, it is possible to extend its capabilities and features. This section describes some of the capabilities and features that could be interesting to look at, if the intent is that MSDLSE shall be commercial off-the-shelf software.

8.3.1 Map interaction

A very interesting aspect of scenario generation is map generation and interaction. MSDLSE offer some ways one might interact with the map; when creating a fresh scenario, being able to import from a bitmap file and vector files (either locally or remotely), and; interaction with the MSDL parameters of the map (such as the coordinates).

However, MSDLSE only allow a static interaction with maps; if one desire to change the map (e.g. the scenario has moved, or expanded, to a different part of the world). Alas, we propose that the map shall become completely interactive, allowing e.g. zooming capabilities, even after the initial map setup. This would allow the simulation to morph, depending on location.

(Of course, as bitmap image are known to become pixilated, it would be wise to restrict such functionality to vector graphic files.)

If any such feature would be added, MSDLSE's scenario management would require slight work over; one map is currently one scenario. If the map is able to morph, depending on the user's needs, the entire world may be a scenario. On the other hand, the user might desire to have multiple scenarios within one "map session". The advantages to this approach are apparent, and deserve no further explanation. However, several important key parts need to be addressed (besides the zooming capabilities etc);

- If one has multiple scenarios in one work space, how do you visually tell one scenario apart from another (from the user's point of view)?
- If one has multiple scenarios in one work space, it is possible that not all scenarios are of the same MSDL version.
- If a MSDL component shall have certain properties in one scenario, the component may have different properties in another scenario (assuming a MSDL component can be added to multiple scenarios).
- If the user selects multiple MSDL components, where the components are not part of the same MSDL scenario, and select to export said components, how shall other relevant MSDL scenario information be treated in the export?

Additionally, to extend map interaction, features could include;

- A 3D-representation of the map (and its components).
- More versatile choice when importing a map.
- Ability to draw arbitrary objects and structures (not necessarily for MSDL per se, but rather as an aid for MSDLSE users).

8.3.2 MSDL components

The most obvious work on MSDLSE is to extend MSDLSE's range of support for MSDL. However, there are other improvements that could be made;

- Ability to view all MSDL components in a scenario.
- More information about MIL-STD-2525B components.
- More information about other MSDL components.
- More elaborate choices for MSDL components' properties
- Further export support, for e.g. Virtual Battlespace or Half-Life.

As well, it is possible in MSDL for units to have equipment and have units be associated with e.g. a tank or airplane. Functionality to display this hierarchy adequately would be beneficial for the MSDLSE user.

Additionally, MSDL will possibly add C-BML (Anon, 2008, Future MSDL focus) support to its base specification, which would be interesting as well for MSDLSE to have.

8.3.3 Other possible improvements

MSDLSE is written in C#, which is mostly for Microsoft's .NET platform. To allow a larger range of users, MSDLSE ought to be ported to e.g. the Mono platform (Icaza, Anguiano, Sánchez, The Mono project), or possibly be written in a different language that is platform independent.

MSDLSE, or even MSDL, is not written to enable real-time change coming from an external source. However, a possible idea is to allow continuous updates from an external source, allowing in-the-field operatives to send information about the surroundings, thus the requirement to update the scenario in MSDLSE.

To stay current with MSDLSE versions, adding an automated updater to MSDLSE would be a minor, but very rewarding, feature.

To allow simpler implementation of newer versions of MSDL or some other specification, MSDLSE could allow a plug-in interface for its scenario modules. This could allow others to develop the “plug-in driver”, and distribute it, without the need for MSDLSE updates. As well, possibly allow the “drivers” to be written in a more human-readable language, than e.g. C++ or C#.
One could as well create a MSDL plug-in for other applications – for other map or scenario rendering software.

Of course, the long-term idea is to have a connection (of sorts) to the program suite(s) that STS provides.

8.4 Interaction with the cooperation thesis

The cooperation thesis is "Serious Games – Integrating games in military training" by John Olsson and Robert Michalski. The cooperation comprises of the serious gaming common reference model and the relationship model for the common reference model presented in section 3.3 and 3.4 respectively.

The reference model was developed in STS' proprietary tool CoDE, and the relationship model in Microsoft Visio, and done in STS' facilities in Helsingborg, Sweden.

The common reference model development consisted of;

- Elicitation of information from the real-time strategy model in this thesis and the first-person shooter reference model.
- Discussion about other possible interesting entities and attributes
- Discussion about general entity abstraction.

The relationship model development consisted of;

- Elicitation of possible entity relationships from the common reference model.
- Discussion about the entity cardinality.
- Discussion about the entity relationship flow direction.
- Discussion about relationship actions.

Everyone participated in the elicitation and discussion process equally, and each discussion point was thoroughly passed after each member had its say. A discussion meeting was postponed if someone was absent. The work flow was natural and well within the time scheme.

Some difficulties were encountered;

- Both theses' respective reference models had to be completed beforehand.
- The general information had to coincide with our real-time strategy common reference model and the cooperation thesis' first-person shooter common reference model.
- Much discussion on almost every entity and its attributes, leading to meetings being prolonged.
- Meetings postponing if someone was absent.

However, the four of us were able to cooperate naturally, so the mentioned difficulties were not as fatal as one might have expected.

9 References

9.1 Official publications

Simulation Interoperability Standards Organization, 2008. Military Scenario Definition Language (MSDL), SISO Inc.

Simulation Interoperability Standards Organization, 2005. Enumeration and Bit Encoded Values for use with protocols for distributed interactive simulation applications, SISO Inc.

Simulation Interoperability Standards Organization, 1999. RPR-FOM, v. 1.0, SISO Inc.

Open Geospatial Consortium, 2008. Open Geospatial Consortium Keyhole Markup Language (OGC-KML), v. 2.2.0, Open Geospatial Consortium.

IEEE, 2008. IEEE-1516 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA), IEEE.

Department Of Defense, 2005. MIL-STD-2525B Common Warfighting Symbology, Department Of Defense.

NATO management board, 2007. Joint Command, Control and Consultation Information Exchange Data Model (JC3IEDM), Edition 3.1B, NATO.

Anon, 2007. Military operations other than war (MOOTW), vol. 1, Anon.

John Olsson and Robert Michalski, 2008. Serious Games – Integrating games in military training, Lund University, LTH.

Hewlett-Packard, Intel and Microsoft, 2006. ECMA-334 C# Language Specification, 4th edition, ECMA.

ISO, 2003. ISO/IEC 10646 Universal Multiple-Octet Coded Character Set, UTF-8, ISO.

World Wide Web Consortium, 2003. Portable Network Graphics (PNG) Specification (Second Edition) Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification, ISO/IEC 15948:2003 (E), World Wide Web Consortium.

Anders Petersson. High Level Architecture, Chalmers Tekniska Högskola, CTH.

Adam Lundgren, Saša Perak, Robert Michalski, Fredrik Ullner, 2007. Final Report – Virtual Battlespace 2, Lund University, LTH.

9.2 Internet

Simulation Interoperability Standards Organization, 2008. MSDL XSD-files.
http://www.sisostds.org/index.php?tg=fileman&idx=get&id=29&gr=Y&path=Products+-+All+Versions%2FMSDL+Specification+Version+1+%28not+released%29%2FBalloting+Package&file=schemaDraftMsdlV1_0.zip
[Accessed Feburary 7th, 2008]

Nick Butler, 2005. A Tree Collection – An implementation of a Tree Collection in C#.
http://www.codeproject.com/KB/recipes/treecollection.aspx
[Accessed March 10th, 2008]

Anon, 2008. Future MSDL focus.
http://discussions.sisostds.org/file.asp?file=MSDLPostVersion10focus%2Edoc
[Accessed April 18th, 2008]

Robert Stone. Education and Training – Serious gaming.
http://www.publicservice.co.uk/pdf/dmj/issue31/DMJ31%202241%20Bob%20Stone%20ATL.pdf
[Accessed April 24th, 2008]

Anon, 2008. Gaming Technologic Impacting Military Training.
http://www.military-training-technology.com/article.cfm?DocID=2268
[Accessed April 24th, 2008]

Anon, 2008. Are Games Here To Stay?
http://www.military-training-technology.com/article.cfm?DocID=2269
[Accessed April 24th, 2008]

Anon. Video Games Grow Up.
http://www.npr.org/templates/story/story.php?storyId=4172753
[Accessed April 24th, 2008]

Anon. Computer Games – Education.
http://wiki.media-culture.org.au/index.php/Computer_Games_-_Education
[Accessed April 24th, 2008]

Lee Copeland, 2001. Extreme Programming controversy.
http://www.computerworld.com/softwaretopics/software/appdev/story/0,10801,66192,00.html
[Accessed May 8th, 2008]

Anon. Geodetic and geocentric coordinates.
http://www.spenvis.oma.be/spenvis/help/background/coortran/coortran.html
[Accessed May 8th, 2008]

Raino Lampinen, 2001. Universal Transverse Mercator and Military Grid Reference System.
http://www.fmnh.helsinki.fi/english/botany/afe/map/utm.htm
[Accessed May 8th, 2008]

Anon, 2007. What is Waterfall?
http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci519580,00.html#
[Accessed May 8th, 2008]

Anon. Benefits of Agile Development.
http://www.versionone.com/Resources/AgileBenefits.asp

[Accessed May 13th, 2008]

Doug Lea, 1995. Namespace definition.
http://g.oswego.edu/dl/WISR93WG/WISR93WG/node1.html
[Accessed May 8th, 2008]

Anon. System.Guid.NewGuid API.
http://msdn.microsoft.com/en-us/library/system.guid.newguid.aspx
[Accessed May 23rd, 2008]

David Helmbold, 2003. Asymptotic growth of functions.
http://www.soe.ucsc.edu/classes/cmps102/Spring04/TantaloAsymp.pdf
[Accessed May 14th, 2008]

Anon. Battlezone.
http://www.recroom-amusements.com/aboutbattlezone.htm
[Accessed May 8th, 2008]

Don Wells, 2006. Extreme programming: A gentle introduction.
http://www.extremeprogramming.org/
[Accessed May 8th, 2008]

Massive Entertainment, World in Conflict.
http://www.worldinconflict.com
[Accessed May 8th, 2008]

Electronic Arts, Command and conquer.
http://www.commandandconquer.com/
[Accessed May 8th, 2008]

Electronic Arts, Battlefield.
http://www.ea.com/official/battlefield/battlefield2/us/
[Accessed May 8th, 2008]

Xmind Ltd, XMIND.
http://www.xmind.org/
[Accessed May 8th, 2008]

Google Inc, Google Earth.
http://earth.google.com/
[Accessed May 8th, 2008]

Simulation Interoperability Standards Organization
http://www.sisostds.org/
[Accessed May 8th, 2008]

Simulation Interoperatbility Standards Organization, Simulation Interoperability Workshop Fall 2008.
http://www.sisostds.org/index.php?tg=articles&idx=More&article=495&topics=124
[Accessed May 8th, 2008]

Simulation Interaoperability Standards Organization, Registration at SISO.
http://www.sisostds.org/index.php?tg=login&cmd=register
[Accessed May 8th, 2008]

World Wide Web Consortium.
http://www.w3.org/
[Accessed May 8th, 2008]

Saab Group and Saab Training Systems.
http://www.saabgroup.com/
[Accessed May 8th, 2008]

One Semi-Automated Forces.
http://www.onesaf.net
[Accessed May 8th, 2008]

One Semi-Automated Forces. OneSAF's MSDL implementation.
http://www.onesaf.net/community/index.php?option=com_content&task=category&sectionid=5&id=18&Itemid=36#18
[Accessed May 8th, 2008]

World Wide Web Consortium. XML standard.
http://www.w3.org/XML/
[Accessed May 8th, 2008]

Microsoft. Microsoft Visio.
http://office.microsoft.com/visio
[Accessed May 9th, 2008]

Valve. Half-Life.
http://www.valvesoftware.com/
[Accessed May 8th, 2008]

Bohemia Interactive. Virtual Battlespace.
http://www.vbs2.com/
[Accessed May 8th, 2008]

Morten Nielsen, 2008. Sharpmap – Open source Mapping engine for the 2.0 Common Language Runtime.
http://www.codeplex.com/SharpMap
[Accessed May 8th, 2008]

Open Geospatial Consortium, Web Map Service.
http://www.opengeospatial.org/standards/wms
[Accessed May 23rd, 2008]

Miguel de Icaza, Jaime Anguiano, Lluis Sánchez. The Mono project.
http://www.mono-project.com/
[Accessed May 26th, 2008]

10 Appendix

10.1 Appendix A

The following XML code describes a scenario in MSDL form, without units or equipment.

```
<MilitaryScenario xmlns="http://www.sisostds.org/Schemas/msdl/v1"
xmlns:modelID="http://www.sisostds.org/schemas/modelID"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ScenarioID>
    <modelID:name>Scenario</modelID:name>
    <modelID:type>BOM</modelID:type>
    <modelID:version>0.1</modelID:version>
    <modelID:modificationDate>2008-04-18</modelID:modificationDate>

<modelID:securityClassification>Unclassified</modelID:securityClassificatio
n>
    <modelID:description>description</modelID:description>
    <modelID:poc>
      <modelID:pocType>Primary_author</modelID:pocType>
      <modelID:pocEmail>project@sts.saab.se</modelID:pocEmail>
    </modelID:poc>
  </ScenarioID>
  <Options>
    <MSDLVersion>1.0</MSDLVersion>
  </Options>
  <Environment>
    <AreaOfInterest>
      <Name>ScenarioMap</Name>
      <UpperRight>
        <GDC>
          <Latitude>180</Latitude>
          <Longitude>180</Longitude>
          <ElevationAGL>0</ElevationAGL>
        </GDC>
      </UpperRight>
      <LowerLeft>
        <GDC>
          <Latitude>-179</Latitude>
          <Longitude>-179</Longitude>
          <ElevationAGL>0</ElevationAGL>
        </GDC>
```

```
      </LowerLeft>
    </AreaOfInterest>
  </Environment>
  <ForceSides>
    <ForceSide>
      <ObjectHandle>0c958265-97ee-4acc-b3ef-
a825dcb5e31c</ObjectHandle>
      <ForceSideName>Civilian</ForceSideName>
      <AllegianceHandle>e642ee19-31a6-43eb-9416-
d350b0a82039</AllegianceHandle>
    </ForceSide>
  </ForceSides>
</MilitaryScenario>
```

10.2 Appendix B

The following XML code describes a scenario with a unit in MSDL form.

```
<MilitaryScenario xmlns="http://www.sisostds.org/Schemas/msdl/v1"
xmlns:modelID="http://www.sisostds.org/schemas/modelID"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ScenarioID>
    <modelID:name>Scenario</modelID:name>
    <modelID:type>BOM</modelID:type>
    <modelID:version>0.1</modelID:version>
    <modelID:modificationDate>2008-04-18</modelID:modificationDate>

<modelID:securityClassification>Unclassified</modelID:securityClassificatio
n>
    <modelID:description>description</modelID:description>
    <modelID:poc>
      <modelID:pocType>Primary_author</modelID:pocType>
      <modelID:pocEmail>project@sts.saab.se</modelID:pocEmail>
    </modelID:poc>
  </ScenarioID>
  <Options>
    <MSDLVersion>1.0</MSDLVersion>
  </Options>
  <Environment>
    <AreaOfInterest>
      <Name>ScenarioMap</Name>
      <UpperRight>
        <GDC>
          <Latitude>180</Latitude>
          <Longitude>180</Longitude>
          <ElevationAGL>0</ElevationAGL>
        </GDC>
      </UpperRight>
      <LowerLeft>
        <GDC>
          <Latitude>-179</Latitude>
          <Longitude>-179</Longitude>
          <ElevationAGL>0</ElevationAGL>
        </GDC>
      </LowerLeft>
    </AreaOfInterest>
  </Environment>
```

```
 <ForceSides>
  <ForceSide>
   <ObjectHandle>0c958265-97ee-4acc-b3ef-
a825dcb5e31c</ObjectHandle>
   <ForceSideName>Civilian</ForceSideName>
   <AllegianceHandle>e642ee19-31a6-43eb-9416-
d350b0a82039</AllegianceHandle>
  </ForceSide>
 </ForceSides>
 <Organizations>
  <Units>
   <Unit>
    <ObjectHandle>9b89f71a-a6d2-4aa5-abc4-
eb6000d7326b</ObjectHandle>
    <SymbolId>S-G----------</SymbolId>
    <Name>a unit</Name>
    <CommunicationNetInstances>
     <CommunicationNetInstance>
      <CommunicationNetType>OTHER</CommunicationNetType>
      <CommunicationNetId>1</CommunicationNetId>
      <CommunicationService>NOS</CommunicationService>
     </CommunicationNetInstance>
    </CommunicationNetInstances>
    <Status>
     <MOPPLevel>LEVEL_0</MOPPLevel>
     <WeaponControlStatus>WEAPONS_FREE</WeaponControlStatus>
    </Status>
    <Disposition>
     <DirectionOfMovement>0</DirectionOfMovement>
     <Speed>0</Speed>
     <Location>
      <GDC>
       <Latitude>10.3678</Latitude>
       <Longitude>61.18505</Longitude>
       <ElevationAGL>0</ElevationAGL>
      </GDC>
     </Location>
     <UnitPosition>
      <FormationPosition>
       <OutOfFormation>1</OutOfFormation>
       <FormationOrder>1</FormationOrder>
      </FormationPosition>
      <OwnFormation>
```

<FormationLocationType>CENTER_OF_MASS</FormationLocationType>
<FormationType>GROUND</FormationType>
<FormationData>
<GroundFormationType>NONE</GroundFormationType>
</FormationData>
</OwnFormation>
</UnitPosition>
</Disposition>
<Relations>
<ForceRelation>
<ForceRelationType>UNIT</ForceRelationType>
<ForceRelationData>
<ForceSideHandle>0c958265-97ee-4acc-b3ef-a825dcb5e31c</ForceSideHandle>
</ForceRelationData>
</ForceRelation>
<SupportRelations>
<SupportRelation>
<SupportedUnitHandle>9b89f71a-a6d2-4aa5-abc4-eb6000d7326b</SupportedUnitHandle>
<SupportType>NONE</SupportType>
<SupportRoleType>NOT_SPECIFIED</SupportRoleType>
</SupportRelation>
</SupportRelations>
<OrganicRelation>
<OrganicRelationData>
<OrganicForceSideHandle>c54cecca-0b94-46bc-8246-3ab964b39afa</OrganicForceSideHandle>
</OrganicRelationData>
</OrganicRelation>
</Relations>
</Unit>
</Units>
</Organizations>
</MilitaryScenario>

10.3 Appendix C

The following table, 11A, lists explanation for abbreviations and terms used in this thesis.

Abbreviations and terms	Explanation
.shp	Shapemap vector file
Agile model	A development model [40]
AI	Artificial intelligence
API	Application Programming Interface
C# runtime	The C# language's environment for running C# applications
Cardinality	Number of elements in a set
Commander	Superior member of a hierarchy
Cooperation thesis	The thesis presented by John Olsson and Robert Michalski
Deserialization	The act of taking an XML and transforming its content to language code
Doctrine	System or policy of teachings
Elicitation	To investigate
Encoding	A specific transformation of data
Enumeration	A list of sorts
Epoch	A significant period of time; used as the time since January 1st, 1970
Exceptions	An unexpected error
Extreme programming	A development model
GDC	Geodetic coordinates
HLA	High Level Architecture
Information model	Abstract model of information flows in a system
ITree	An open source API
JC3IEDM	A standard for joint command references and connections
km/h	Measure of speed; kilometers per hour
Markup language	An artificial language that structures text in a specific manner
Methodology	A set of methods and practices
Milestone	A significant event
MIL-STD-2525B	MIL-STD-2525B Common Warfighting Symbology
MOOTW	Military Operations other than war
MSDL	Military Scenario Definition Language
Namespace	A scoping construct to subdivide the set of names and their visibility in a system
nmi/h	Measure of speed; Nautical miles / hour

O(n), (O1)	Time complexity [44]
OGC-KML	An extension to the XML language
Overlays	A surface layer or design
Panacea	Cure for everything
Pixilated / Pixels	The smallest part of an image
Placemark	A mark at a specific place or location
PNG	Image compressed in the PNG format
Proprietary	Belonging to a proprietor
Quirk	Unusual behavior
Refactoring	Restructuring of code
SISO	Simulation Interoperability Standards Organization
STS	Saab Training Systems
Symbology	Use of symbols
Validated	Match a set of requirements
Waterfall model	A development model
Vector	Mathematical geometrical data to represent images
WMS server	Web Map Service server
XML	Extensible Markup Language
XSD	A validation technique for XML files
MSDLSE	Military Scenario Definition Language Scenario Editor
CRM	Common reference model
SSGCRM	Strategy Serious Gaming Common Reference Model
SGCRM	Serious Gaming Common Reference Model

Table 11A – Abbreviations and terms

10.4 Appendix D

The following XML code describes a unit at a certain coordinate in OGC-KML.

```
<kml xmlns="http://earth.google.com/kml/2.2">
  <Document>
    <Style id="S-G----------">
      <IconStyle>
        <Icon>
<href>C:\Symbols\2525B\appendixa_png_svg\appendixa_png_svg\warfighting_symbols\ground_track\0.sugp----------.png</href>
        </Icon>
      </IconStyle>
    </Style>
    <Placemark>
      <name>Unit</name>
                <description>Unit description</description>
      <styleUrl>#S-G----------</styleUrl>
      <Point>
        <coordinates>96.9572982788086,87.4306335449219,0</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

10.5 Appendix E

Software Requirements Specification for MSDL Scenario Editor

Fredrik Ullner, Adam Lundgren

2008-03-17

Abstract

This document describes the requirement specification for the Military Scenario Definition Language Scenario Editor (MSDLSE), made by Fredrik Ullner and Adam Lundgren. This document assumes background of the Military Scenario Definition Language.

The requirements are written in plain text English and as functional requirements.

The document is divided in two parts; General requirements (GR) and MSDL related requirements (MR). Each requirement is assigned a number, with its corresponding letter identifier. Any references should be made with the letter and number combination.

Table of content

1. General Requirements – GR
These requirements describe general features or other requirements related to MSDLSE.

GR1. MSDL tree structure
The MSDL specification describes MSDL as a tree of nodes with attributes and further child nodes. The MSDLSE should store each MSDL node in a similar way.

GR2. MIL2525B icon set
The icon set used to display the MSDL items shall be icons from the MIL2525B specification.

GR3. Save and Open dialog boxes
Before a user imports or exports items, dialog boxes should appear at the user's selection.

GR4. Import of bitmap image
Importing a bitmap image as the scenario background, allowing context for the MSDL items, should be possible.

GR5. Export of bitmap image
Exporting the bitmap image that is used as the scenario, background should be possible.

GR6. Printing bitmap image
Printing the bitmap image.

GR7. Settings file
Settings, such as log file location or other preferences regarding the totality of MSDLSE, should be able to be read, parsed and exported to and from XML files.

GR8. Error notification and handling
Error notification when user inputs incorrect values, when MSDLSE crashes or other errors shall be reported either directly to the user or at least logged in .log files.

GR9. Multiple workspaces
The user should be able to work with multiple scenarios at the same time (regardless if it might be confusing for them). This will be presented in the form of tabs in the MSDLSE.

GR10. Moving an object from one workspace to another
While copying within one workspace should be possible (see MR7), copying between two workspaces (tabs) should also be possible.

GR11. Help and copyright information
The MSDLSE shall present tips, guides or other helpful information for the user when using the application. The user shall also be made aware of any copyright information regarding MSDLSE.

2. Military Scenario Definition Language Requirements – MR
These requirements describe features and the MSDLSE relation to the MSDL specification.

MR1. Support part of MSDL specification
The MSDL specification is too large to wholly implement. As such, only certain parts are implemented; Those that the specification require, Organizations (for units and equipment) and Environment (for the environment area).

MR2. Import of MSDL XML file
Importing a MSDL XML file and having the application arrange a MSDL tree structure, according to the information. Note that this is not part of the Acceptance Requirements for Exjob1, and should only be done if time is available.

MR3. Export of MSDL XML file
Exporting the MSDL tree as a XML file, according to the MSDL specification.

MR4. Validation of MSDL XML file
An exported or imported MSDL XML file should be validated against the MSDL XSD files, to make sure the XML file is correct.

MR5. Default attributes for MSDL components
Each MSDL component's attributes that can have default values, will have so.

MR6. Inserting a "MSDL item" to a map
The interface shall allow MSDL components, or items, to be display in a previously imported bitmap image. When an item is inserted, they should be inserted in the internal MSDL tree structure.

MR7. Select
All MSDL items in a bitmap image shall be possible to selectable.
MR8. Copy
ALL MSDL items in a bitmap image shall be possible to copy.

MR9. Remove.
All MSDL items in a bitmap shall be possible to remove.

MR10. Move
All MSDL items in a bitmap image shall be possible to move.

MR11. Properties for "MSDL items"
When selecting a MSDL item, the item's properties should be configurable through the interface of MSDLSE.

MR12. "MSDL items" as commentary
The user should be able to determine, at run time, if a MSDL item should be exported to XML. This option can allow items that are only there for informational purposes only, and therefore should not be processed and used in the XML.

10.6 Appendix F

The following script code describes inserting a unit and a force in a Virtual Battlespace 2 script.

```
_groupCivilian = createGroup Civilian;
"SolderW" createUnit [[2931.93717277487, 560.498220640569],
_groupCivilian];
```