# Serious Games

## - Integrating games in military training

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

**LTH School of Engineering at Campus Helsingborg**
**Computer Science**

Bachelor thesis:
John Olsson
Robert Michalski

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 Games – Integrating games in military training.

Serious games are computer games used for various training in virtual environments. This can in many cases reduce costs, time and gain better results over ordinary training. Complex training scenarios which would be practically infeasible to create in reality can be constructed in a virtual environment.

Saab Training Systems have developed an integration platform called WISE (Widely Integrated System Environment). Any application can be integrated and communicate with other applications using WISE. Computer games are very attractive in the field of training due to their immersive realism, relatively low cost and scalability.

The questions discussed in this thesis are; how can computer games be integrated with military training systems? What information does FPS and strategy games have in common? How can a scenario in the MSDL-format (Military Scenario Definition Language) be imported into a game?

Using WISE as an integration platform, computer games can be connected through the use of a game plug-in and a WISE-driver combined with a common game information model. Scenarios described using the MSDL language, can be imported either through translation to game specific script, if the game supports scripting or through the WISE-driver and game plug-in. A generic WISE-driver for games, a VBS2 plug-in and a Half-Life 2 Empires plug-in have been implemented as part of this thesis.

Keywords: Serious Games, Saab Training Systems, Half-Life 2,VBS2, WISE, MSDL, military training

## Sammanfattning

Serious Games – Integrering av datorspel i militär träning

Serious games är datorspel som används för olika typer av träning i virtuella miljöer. Det kan i många fall leda till reducerade kostnader, sparad tid och ge bättre resultat än vanlig träning. Komplexa träningsscenarion som inte är praktiskt genomförbara att skapa i verkligheten kan konstrueras i en virtuell värld.

Saab Training Systems har utvecklat en integrationsplatform, kallad WISE (Widely Integrated System Environment). Vilken applikation som helst kan integreras med andra applikationer genom WISE. Datorspel är väldigt attraktiva i träningssyfte eftersom de är väldigt verklighetstrogna, relativt billiga och skalbara.

Frågorna som skall belysas i examensarbetet är; hur kan datorspel integreras med militära träningssystem? Vilken information är gemensam för första persons skjut och strategi spel? Hur kan ett scenario i MSDL-format (Military Scenario Definition Language) importeras i ett spel?

Datorspel kan integreras med andra system genom att använda WISE som gemensam integrationsplatform. Detta uppnås genom att skapa ett spel plug-in och en WISE-drivrutin och använda en generisk spelinformationsmodell.

Scenarion i MSDL-format kan importeras i spel genom att översättas till spelspecifikt skript om spelet stödjer skripting eller genom WISE-drivrutinen och spelets plug-in. En generisk WISE-drivrutin för spel, ett plug-in för VBS2 och ett plug-in för Half-Life 2 Empires har utvecklats som en del av detta examensarbete.



Nyckelord: Serious Games, Saab Training Systems, Half-Life 2, VBS2, WISE, MSDL, militär träning

## Foreword

This thesis consists of this document, development of two plug-ins and a driver for the WISE platform. It is written by John Olsson and Robert Michalski and is carried out at Lund University, LTH School of engineering. The work is done for Saab Training Systems at the office in Helsingborg.

We would like to thank the examiner Christin Lindholm and the technical supervisors, Jakob Blomberg and Niklas Andersson. Thanks go out to employees at Saab Training Systems in Helsingborg, Per Gustavsson for feedback and finally to Adam Lundgren and Fredrik Ullner for their cooperation.

# Table of contents

# 1 Background

This is a Bachelor thesis written by John Olsson and Robert Michalski for Saab Training Systems, hereafter referred to as STS, during quarter one and two of 2008. STS main field of business is military training equipment and the division in Helsingborg specialises in integrating different types of equipment and training systems together. The information presented in this chapter comes from an internal oral presentation by the supervisor of this project (Jakob Blomberg, 2008).

## 1.1 Military training

The classic type of field training is hundreds or thousands of soldiers and vehicles training together in a battlefield. There are other forms of training such as simulations and serious games, where soldiers and commanders can sit at a PC or mock-ups of vehicles and train tactics, procedures or visualize scenarios. Using virtual environments can be as real as being out on the field and yield similar results at a much lower price. All soldiers and equipment do not have to be in the same place and thus do not need to be transported, which would save time and money.

## 1.2 Problems with connecting different systems

The military is focusing more and more on joint operations and collaboration across nations. There are several providers of military training equipment, simulators and serious games, each use their own proprietary technology. Different military powers or forces acquire their training equipment from several different vendors. To train joint operations, equipment from different forces shall be able to communicate with each other. The military industry has agreed on certain standards. There are many additional standards in development and vendors create interfaces that implement some of those standards. However, protocols have limitations and are not well suited for every application and possible use. Many types of equipment, simulators and software do not support the same standards and protocols and thus cannot really communicate with each other.

It would require tremendous amounts of resources and time from every vendor to modify their products to work with every other vendors' products. Many complications, ambiguities and other problems can arise during the process. If every system would be modified to connect directly to all other systems it would require n(n-1) modifications, where n is the number of systems connected [Figure 1].

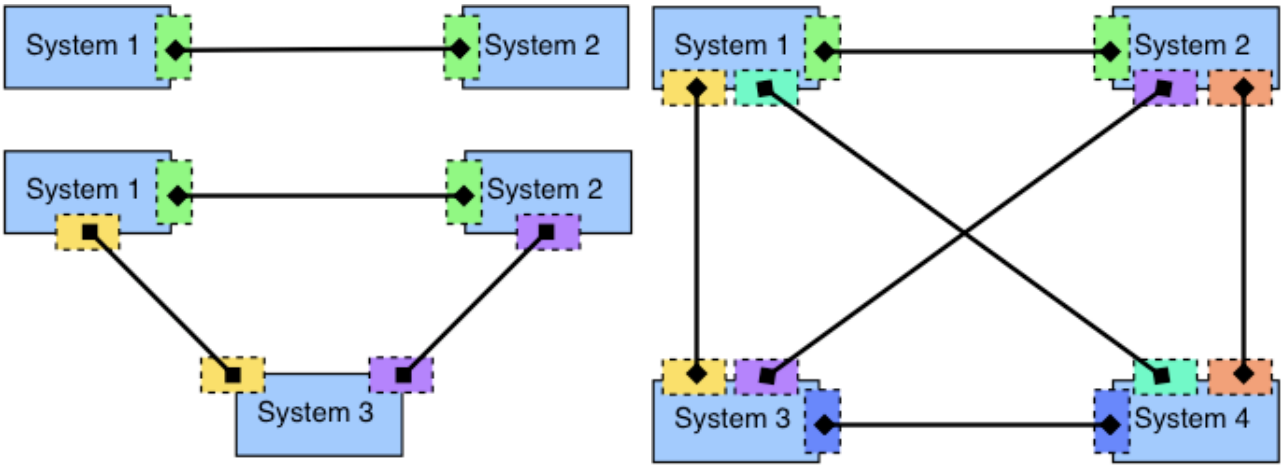


**Figure 1 - Systems modified to communicate with each other. Dotted boxes are modifications made to the systems.**

There are several standardized architectures and protocols, for instance DIS (Distributed Interactive Simulation) (Institute of Electrical and Electronics Engineers, 1998, IEEE 1278.1A-1998 - Standard for Distributed Interactive Simulation - Application protocols) and HLA (High Level Architecture) (Institute of Electrical and Electronics Engineers, 2000, IEEE 1516-2000 - Standard for Modeling and Simulation High Level Architecture - Framework and Rules), which are used to connect different systems together. Standardized architectures and protocols help reduce the number of modifications needed to connect systems together, to only one modification for each new system that is added [Figure 2].

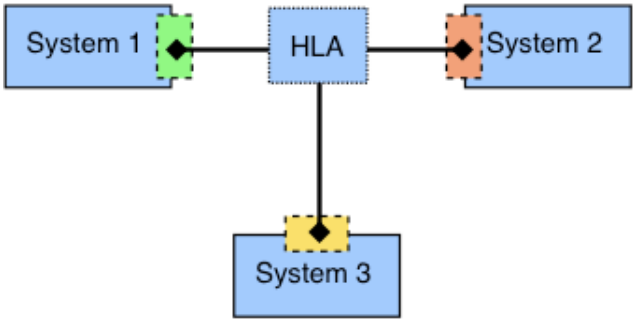


**Figure 2 - Systems connected using HLA. Dotted boxes are modifications made to the systems.**

## 1.3 STS' solution to connecting different systems

One solution to the problem of connecting systems that were not designed to work together is to let all the systems remain as they are and make a platform that each system can connect to using a native or standard protocol that it supports. The platform then translates information from each systems' format to the other systems' native formats. This way only the platform that everybody connects through needs to be modified to support a new system or

protocol. STS have developed a platform which can do just that, called WISE. The WISE platform can be extended to handle any system, such as a hardware simulator, training equipment from many different manufacturers, software that use HLA, DIS or its own protocol. Any system can be integrated with another system using WISE. To integrate a system with WISE an information model and a driver are required. The information model specifies which information the system will be able to share and how it will be structured. The driver enables the system to communicate with WISE using the system's own native protocol. If a system does not have built in support for communication an additional plug-in has to be created. WISE already has support for several systems and now STS want to add support for serious games and consumer games [Figure 3].

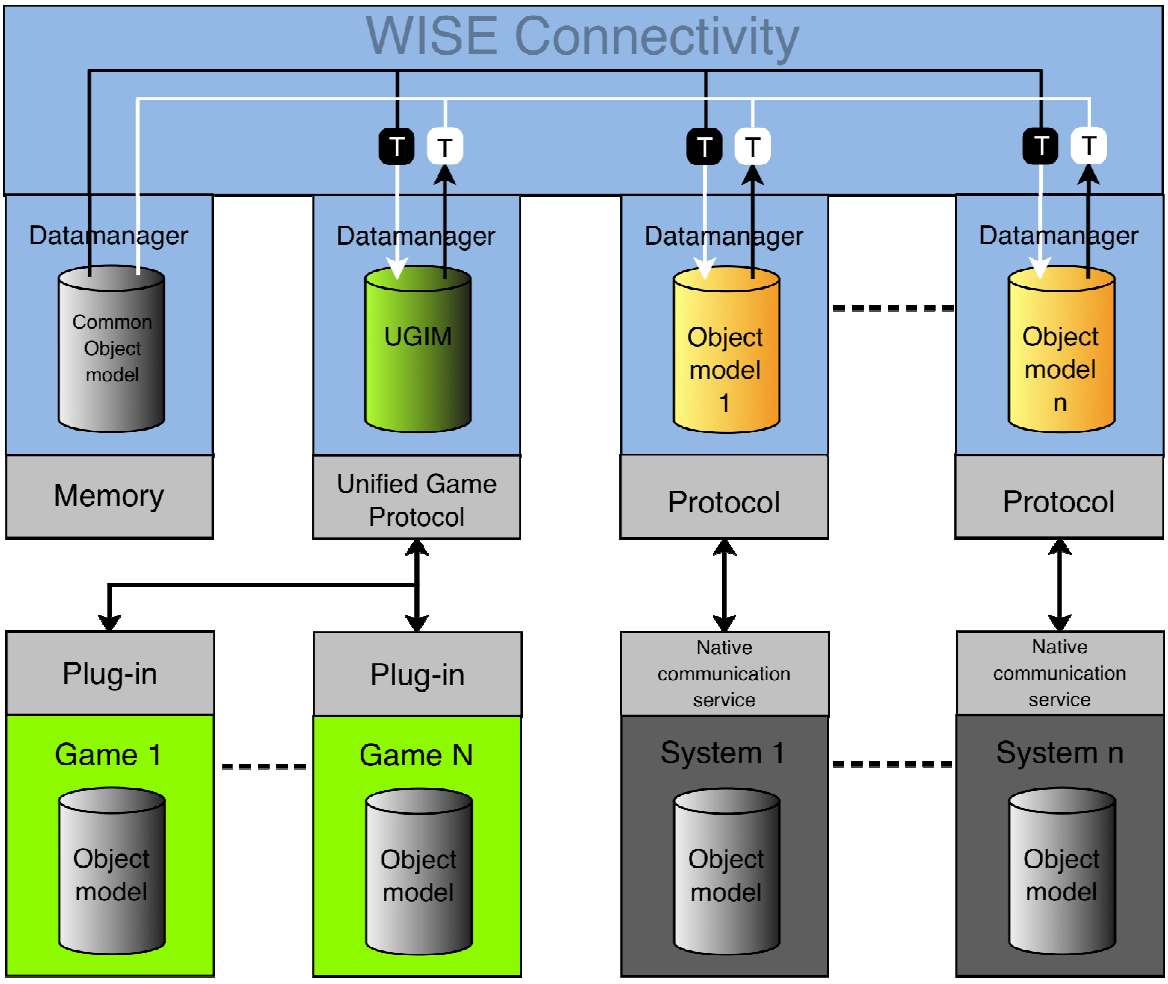


**Figure 3 - Different systems connected to WISE using their own native protocols. The games connect using one universal computer game driver (UGD) and separate game plug-ins, GPIs. Each vertical block attached to WISE Connectivity is a WISE driver. The boxes with a T in them symbolize data transformations.**

There are three problems when integrating a computer game; every computer game supports different features and it is not always possible to retrieve the same information from different games; the way to retrieve information

between different games varies; there are usually no native communication services that can be used to retrieve information. A plug-in can be used to retrieve desired information from each game. A driver, which uses an information model that supports the most common features in FPS type of games, inserts the information into WISE.

## 1.4 Why serious games?

Specially designed equipment and software are very expensive. The visual effects and virtual environments in consumer computer games have become very realistic and can run on generic PCs that are relatively cheap. This makes computer games well suited for military simulation and training. The visual quality is more than good enough already and is only going to improve even more over time. Costs can be reduced by a factor of ten or even a hundred compared to buying specially developed equipment and simulators.

Serious gaming is a concept of using computer games and applications for education and training purposes. Some serious games are specially developed for the military and contain features requested by the military customers who purchase them. They are meant to be sold in relatively low volume, a few hundred to a few thousands copies. Serious games are not available for purchase by civilian individuals and are about a hundred times more expensive than traditional consumer games. They are still ten or a hundred times cheaper than specially developed simulators and other similar training equipment.

There are many advantages using serious games for training, education and simulation:

- Complex scenarios, which are practically impossible to setup in real life can be constructed in a virtual environment. For example a forest fire can be simulated in a virtual environment and used for training, instead of lighting a part of a real forest on fire.
- Virtual environments are extremely flexible; buildings can be created, destroyed, altered or set on fire; it is possible to change the location of the training course to anywhere in the world with a few commands.
- Scalability, add more objects such as vehicles, buildings or nature elements at no extra expense. It does not matter if one, two or ten buildings are blown up in a virtual environment. Want larger, more complex environments, add more computing power.

Serious gaming can not replace conventional training entirely. However, used properly it can enhance training results.

## 1.5 Could consumer games be used for serious gaming?

Consumer games are relatively cheap per license and meant to be sold in high volume in the order of ten thousand to millions of copies at a $50-100 price point. New games are released rapidly; each release makes physics, sound, environment behaviour and graphics more real and lifelike. There are also huge communities where fans of the games dedicate their free time creating new content and modifying the way games behave, essentially creating new games using the same engine.

The innovation and work by the computer game modification communities are a huge untapped resource that could be used to enhance military training. If consumer games could be adapted and used for military training it could bring down costs by a factor of ten or a hundred from specially developed serious games and a factor thousand or more from simulators and special training equipment.

## 2 Problem description

Serious gaming is a relatively new way of military training using computer games, leveraging the low cost and easy availability of the PC and the fast technological pace of the computer gaming industry. A lot of the exercises and training otherwise conducted using classic military field training could be effectively performed sitting at a computer practicing in a virtual environment. There are several advantages to training in virtual environments over training the classical way such as less transportation and equipment, more flexible and easier to manage.

STS, a leading developer, maker and provider of military training equipment, has a new platform that is specialized in connecting system together. Now they want to integrate computer games with other applications. This thesis is about laying some of the groundwork of WISE future computer game support through the completion of three assignments, with emphasis on the first two assignments which also make up the acceptance requirements (Jakob Blomberg, 2008).

The first assignment is an examination of what information computer games of FPS type have in common and use it to create an information model that is well suited for all FPS type of games.
The second assignment is to integrate a FPS type of game using the WISE platform [Figure 3].
The third and final assignment is to examine the possibilities to load a scripted scenario based on MSDL into a computer game. This assignment shall be done in collaboration with the thesis called "Lessons learned from implementing a MSDL Scenario Editor – A tool for the serious gaming community", hereafter referred to as the collaboration thesis (Lundgren & Ullner, 2008).

The following questions shall be answered in this thesis:

- How can computer games be integrated in military training using the WISE platform?
- What information does FPS and strategy games have in common?
- How can a scenario script in the MSDL format be imported into a computer game?

## 2.1 Goals

The goals are divided into two subchapters, subchapter one is about STS goal with serious gaming. It has been included to provide a better understanding of design decisions affecting the information model and many of its attributes. Subchapter 2 describes the goals of this thesis.

### 2.1.1 STS serious gaming project goal

STS goal is to make their system compatible with as many other systems as possible. This would enable them to sell their products to a broader customer base e.g. to customers who use products from other vendors. There are many different computer game companies who compete with each other. The result is that the computer game industry keeps a very high pace, there is lots of innovation and consumer games are cheap compared to software which is specially developed for military applications. There are computer game modification communities who put in a lot of voluntary work by creating maps, changing the functionality of games and providing many new, innovative ideas.

STS wants to leverage the computer game industry to enhance their own products by using computer games for military and civilian training. By making sure that their product works with as many computer games and simulation systems as possible, they can offer their clients a product that works with the clients' equipment and with the future systems their clients want.

STS ultimate goal in the scope of this thesis is to integrate games to work like this; Player 1 is playing game A and player 2 is playing game B. Player 1 can interact with player 2 inside game A and player 2 can interact with player 1 inside game B just like both players would be in the same game.

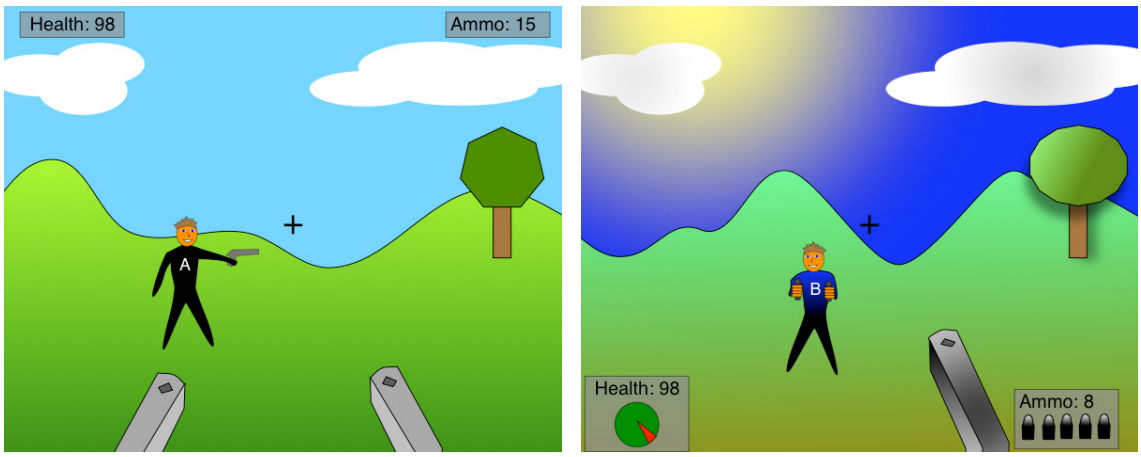


**Figure 4 - Player 1 can see player 2 in game A, to the left, and player 2 can see player 1 in game B, to the right.**

Game A and game B are FPS games and should be able to interact with game C which is a strategy game.

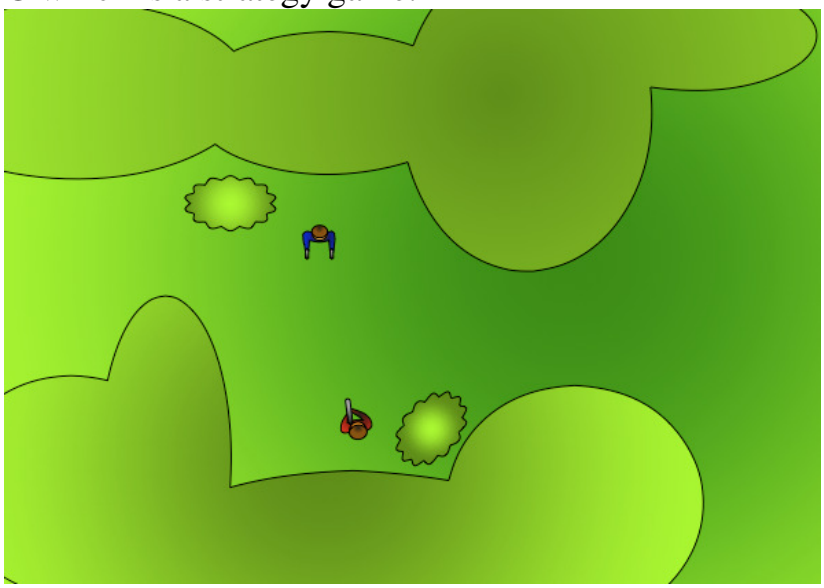

**Figure 5 - Player C can see player A and B inside game C, which is a strategy based game.**

It should also be possible to use visualize a real life training session in game A, B and C and see the units interact in all of the games.

Another goal is to simplify the preparations for training sessions, by using a standard called MSDL which is based on XML. The system should be able to import a scenario file described using MSDL and create training scenarios in all games and systems connected. This saves work because then it would no longer be necessary to create the same scenario in every game and systems native format.

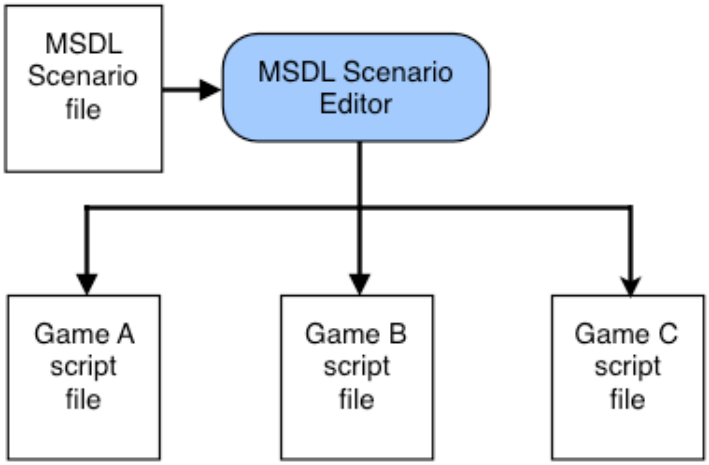


**Figure 6 - Conversion of a MSDL scenario file to other games native script formats.**

It should also be possible to create a scenario in one of the connected games or systems scenario editor and convert the native game script format to the MSDL format. Then it would be possible to use any of the connected games' or systems' scenario editors to create scenarios for all games and systems. It would be a very flexible way to create scenarios, because the best suited scenario editor for each client and situation could be used. It would also be possible to convert from one game script format to another.

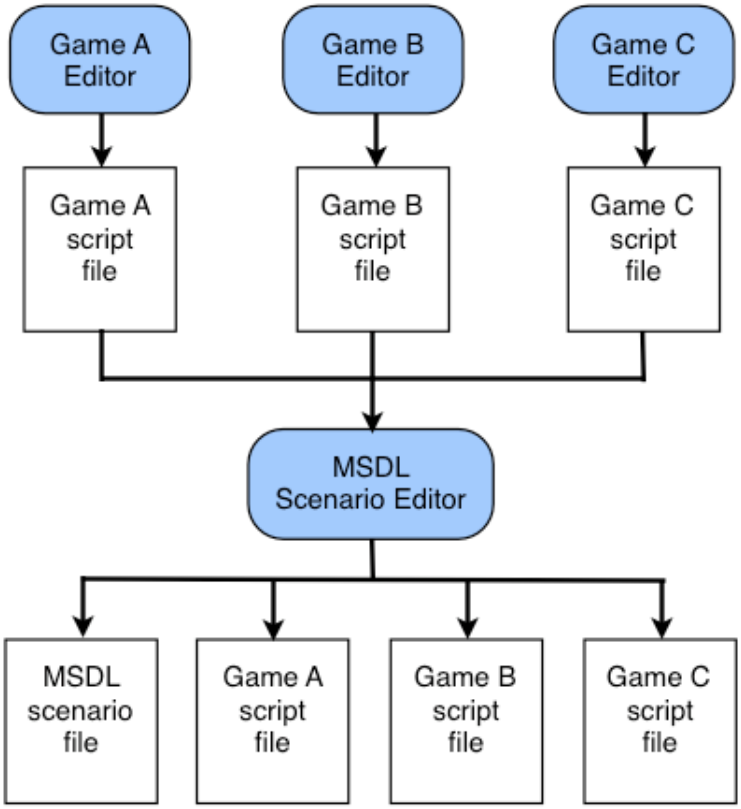


**Figure 7 - Conversion from any game script to any other game script or MSDL.**

For instance, a scenario is created in game As' editor and is converted to game B script, then it can be loaded into game B or opened in game Bs' scenario editor.

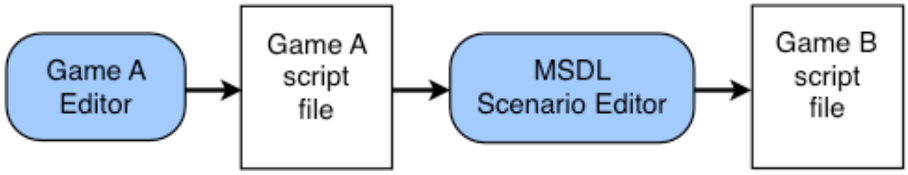


**Figure 8 - Conversion from one game script format to another.**

### 2.1.2 The goal of this thesis

The first goal of this thesis, which has the highest priority, is to create a communication path between a computer game and the WISE platform. It shall be possible to view in-game events in STS software, for instance, player movements, who is shooting at who etc. The second goal is to create a generic information model for FPS type of games. The information model shall be generic enough to be used with any FPS type of game and it should be possible to use it with strategy type of games.
The third goal is to load a scripted scenario, created in a visual MSDL editor described in the collaboration thesis (Lundgren & Ullner, 2008). There is also a bonus goal which is to become co-writers in a paper about the MSDL standard that is going to be presented on a military conference in the US. The most important thing is to fulfil the acceptance requirements and pass this course.

## 2.2 Scope

What game is this thesis focused on? There is a serious game that the entire military training industry is very interested in, it is already widespread and has many good features. The game is Virtual Battlespace 2, (VBS2) by Bohemia Interactive (Anon, 2008, Virtual Battle Space). There is one large problem with VBS2, there did not exists any API in quarter 1 of 2008. However, Bohemia released an updated version of VBS2 with plug-in support in quarter 2 of 2008. Therefore STS searched for alternatives and found Half-Life 2 by Valve (Anon, 2008, The Orange Box) which is suitable because it has a good SDK (Software Development Kit), has a large and very active game modification community and there is a modification which combines FPS type of gameplay with aspects from strategy games called Empires (Anon, 2008, Empiresmod.com). A part of the research in this thesis consists of examining alternative games that could be used instead of Half-Life 2. Games examined were Call of Duty 4 (Anon, 2008, CALL OF DUTY) and Battlefield 2 (Anon, 2006, EA ::Battlefield 2). The examination found that Half-Life 2 was the best choice because it has well documented APIs, a good SDK and huge user communities; therefore Half-Life 2 shall be integrated.

### 2.2.1 Information model

The information model shall contain data that exists in most FPS type of games and enough information to be able to interact sufficiently with players in other games, even strategy based ones. Not all features in every game must be supported; only the most generic actions and features that exist in most of the games have to be supported. It shall be optional to implement certain features that exist in some games but not in others.

Minimal list of actions and features that must be supported by the information model:

- Movement
  - Basic movement: front, back, left, right
  - Jump
  - Crouch
  - Possible to expand with more actions
- Weapon handling
  - Fire
  - Reload
- Life or health
  - Restore health
  - Decrease health
- Communication
  - Text messages
  - Objectives
- Scenario creation, setting start-values for
  - Players
  - Buildings
  - Areas of Effect, minefields, weather, triggers.
  - Small objects, crates, barrels etc.

### 2.2.2 Driver and server side plug-in

Enough functionality shall be implemented in the driver and plug-in to allow them to do the following [Figure 12 - Overview of plug-in and driver architecture.]:

- The plug-in shall be able to send information from the computer game to the driver.
- The driver shall be able to receive information from the plug-in and insert the information into WISE.
- The driver shall be able to send information from the WISE platform to the plug-in.

- The plug-in shall be able to receive information from the driver and insert it into the computer game.

The following is going to be implemented in driver and plug-in:

- Player information
  - ID
  - Name
  - Position
  - Health
- Detonation event (when an unit is hit in the game)
  - Source ID
  - Name
  - Health
- Fire event (when an unit fires a weapon)
  - Source ID
  - Name

### 2.2.3 Import of MSDL file

Import of a MSDL scenario file will be done in conjunction with the collaboration thesis (Lundgren & Ullner, 2008). The scripting languages of the chosen computer game shall be explored and a few simple things shall be scripted, e.g. creating a unit. A simple scenario shall be created using the visual MSDL editor, which is described in the collaboration thesis (Lundgren & Ullner, 2008), and exported, to a game script file which shall be loaded by the integrated computer game and run successfully. This way a scenario could be created in the game from a MSDL file.

## 2.3 Limitations

This thesis is not supposed to make any finished product that can be used in a commercial application or system. All products are on a proof-of-concept scale and are only intended to provide information and examples that can be implemented and incorporated into future projects or extended in future theses. The implementation of driver and plug-in can be used in demonstrations but shall probably not be sold to customers or used in critical systems, without additional functionality and before conducting more thorough testing.

# 3 Work methods

This chapter describes the work methods and procedures used to find solutions and answer the questions of this thesis.

## 3.1 Algorithm

This thesis consists of three phases. Phase 1 and 2 are the most important and must be completed to fulfil the acceptance requirements. Phase 2 and 3 will overlap to some extent. During phase 1, the information model shall be developed and used in phase 2. Phase 2 will result in a plug-in and a driver that shall provide a communication path between a computer game and the WISE platform. The length and extent of phase 3 is determined by how much time is left to deadline.

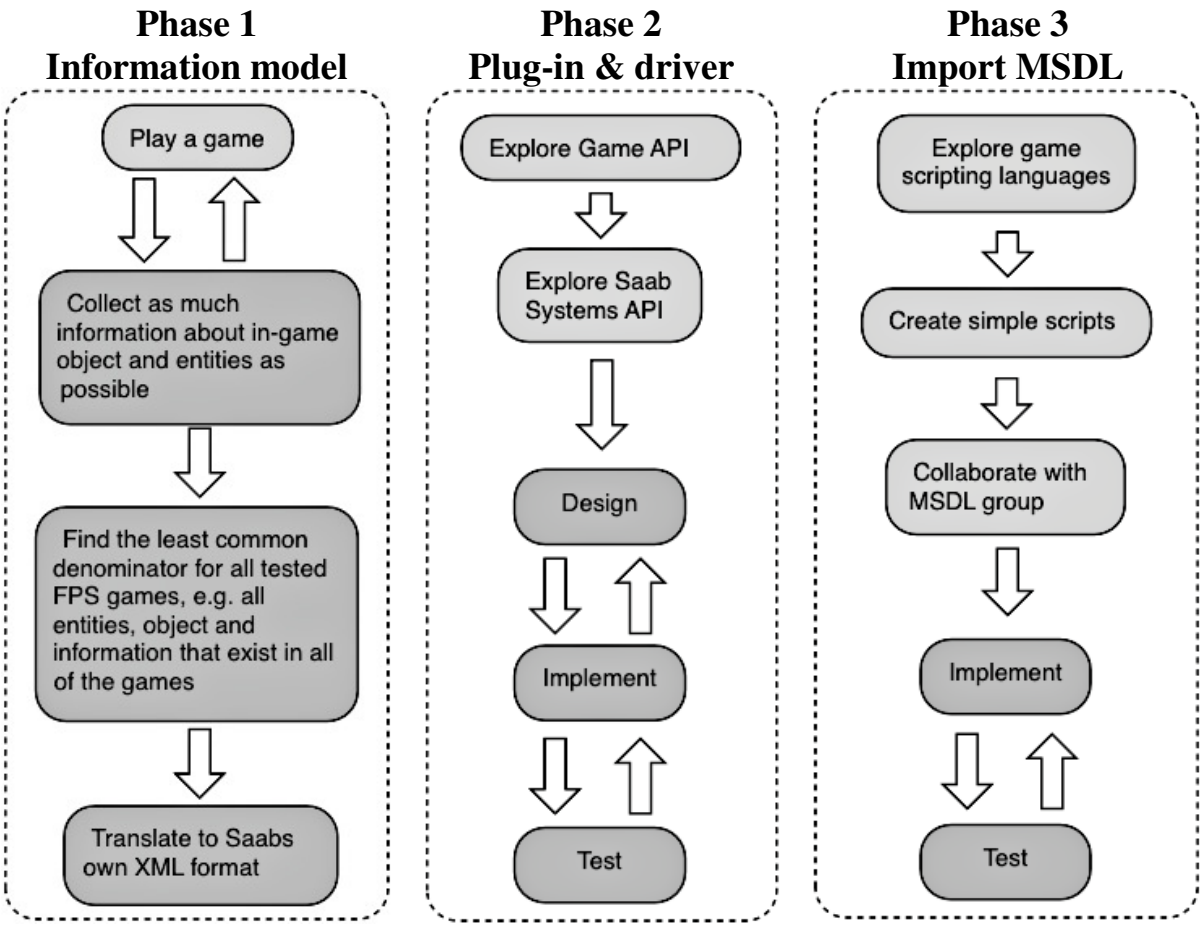


**Figure 9 - The phases of this bachelor thesis.**

### 3.1.1 Phase 1 – Information model

Information is gathered by playing different FPS type of games, such as Virtual Battle Space 2, Half-Life 2 with a few different modifications, Call of Duty 4 and Battlefield 2. While playing a game observations are written down. All the collected information is structured and a draft of the model is created.

The information model is revised several times, restructured and simplified each time. An E/R-diagram for the entities in the information model is also created to show how entities relate to each other. Finally the information model is translated to STS own XML based format using STS proprietary tools.

### 3.1.2 Phase 2 – Plug-in & driver

A game engine is chosen, based on if there is a SDK available for the game and how well documented it is. The SDK of the chosen game is explored and evaluated to determine if it is possible to do all the things needed to fulfil the acceptance requirements by trying to implement simple things such as getting the position of a unit. Then the design process is started. Implementation and testing is executed in parallel. The design is revised and improved during implementation and testing. Only the most important functionality is implemented until deadline. If implementation is finished before deadline, the next phase is entered earlier.

### 3.1.3 Phase 3 – Import MSDL

The game scripting languages are explored and simple scripts are created. The specification of how to script a few simple things, such as creating a unit, is provided to the group working on the collaboration thesis (Lundgren & Ullner, 2008). Export functionality for the game specific script is incorporated into the visual MSDL scenario editor application and a simple scenario script is created and exported to a game script. The game script file is loaded by the computer game and run. Scripts are tested, revised and adjusted until they work properly.

## 3.2 Time plan

The time plan is divided into three phases which overlap. The final report is written during the entire span of the thesis. Time is displayed in project weeks at the top of the table and as calendar weeks at the bottom. This thesis spans over 19 weeks because of shorter workdays and some days off. There are 4 checkpoints that are supposed to simplify keeping track of the project and presenting results to the supervisor. At each checkpoint the time plan is revised and a risk analysis is performed.

The following is performed at each checkpoint:

- Checkpoint 1:
  - Present information model to supervisor
- Checkpoint 2:
  - Present to supervisor how far integration has come and what has been implemented.
  - Present to supervisor the comparison of game tools [Appendix 9.1].
- Checkpoint 3:
  - Demonstrate how the integration works and the things that have been implemented so far.
- Checkpoint 4:
  - Hand in thesis, code and application.
  - Start working on presentation.

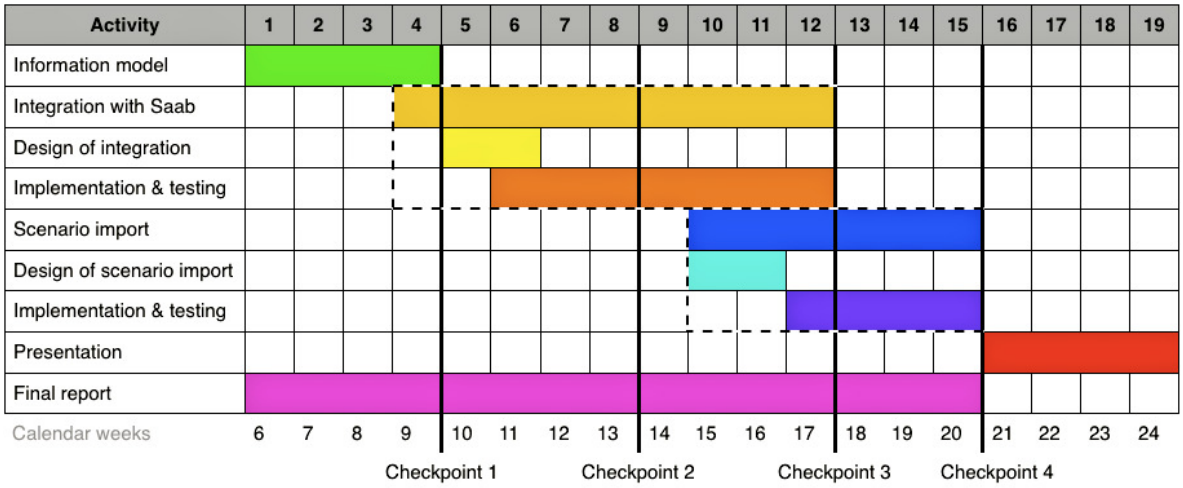


**Figure 10 - First time plan.**

Things did not turn out as expected, there were several delays and the time plan had to be changed. Phase 1, which is the green bar, was extended by two weeks because the information model took longer than expected to complete. Phase 2 was also moved forward by two weeks, because some network ports that were needed to run the game to be integrated where blocked. A workaround in the form of a 3G modem was ordered.

In the meantime STS requested a comparison of game developer tools, which resulted in chapter 4.4. The 3G modem finally arrived and provided full and unrestricted access to the Internet. However, the 3G modem worked less than optimally and often dropped the connection to the Internet, therefore the progress was slow. A few weeks later a 3G router was provided which worked perfectly, work progressed faster. The design of integration and MSDL scenario import were also extended, because of limited knowledge and ambiguous documentation of the games, a trial and error approach was

preferred. Checkpoint 4 was moved two weeks forward to provide more time to complete phase 2, 3 and the final report.

Checkpoint 4 was renamed to Finish, because it is the deadline of this thesis. Phase 2 was extended 5 weeks because it was the top priority and the main focus of the entire thesis. The time to create a presentation of the thesis was shortened to provide time to work on the thesis. There was a big unexpected event that affected the thesis one week prior to checkpoint 3. STS got access to the VBS2 VTK (Virtual Toolkit) and an integration point to VBS2 and ordered immediate cease of development of the HL2 (Half-Life 2) Empires integration and to shift focus to integration with VBS2.

Luckily it was easier to integrate with VBS2 than with HL2 and work progressed rapidly. STS was supposed to provide a course in driver development before checkpoint 3, but it was delayed 5 weeks into the middle between checkpoint 3 and the deadline of the thesis, which means that there was only a short amount of time for driver development.

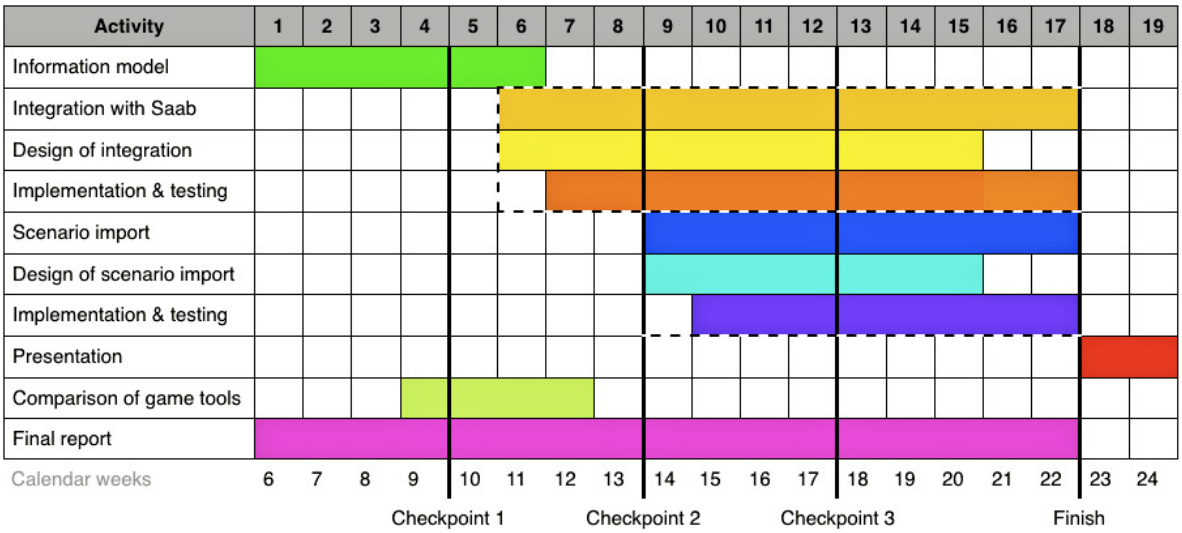


**Figure 11 - Final time plan.**

## 3.3 Proposed solution

There are several possible solutions to the assignments this thesis is meant to elaborate on. This chapter consist of several possible solutions to each assignment together with an explanation of why a particular solution was favoured.

### 3.3.1 What information do different FPS and strategy games have in common?

The information required to integrate FPS games with strategy games depends on what game features shall be supported. FPS games and strategy games both support basic units, such as players and vehicles. The units shall be able to move, fire at each other and report what status they are in. There are many more features that could be supported, such as buildings, small objects, nature objects, animals, weather and many additional features. The two following solutions were considered. The expanded information model was chosen because STS wanted the model to support as much as possible.

#### *3.3.1.1 Small information model that covers essential features*

A small model that provides basic features is fast, simple to implement and can be tested thoroughly in a short amount of time. The small model shall be possible to use with every FPS game. It shall be possible to specify own types and own values to be able to use the model with future games that may support new future features. This makes the model a lot more flexible and expandable. It is very simple to expand by adding more attributes to the entities in the model new functionality can be added.

#### *3.3.1.2 Expanded information model*

A large model could be developed by extending the small model in the previous chapter. It shall support a lot of features, and it is especially useful if different games that are connected together shall be consistent and coherent e.g. look and feel the same. Some attributes could be required and the rest optional which would give the model flexibility and scalability. Network load would increase but features in newer games would not be unused. The structures used to send information over the network would be the same in all games and have the same size. Games that do not support certain features would just enter default value into those attributes in the structure and useful information into the rest of the attributes. The model could also be used with other types of computer games than FPS, for instance strategy and flight simulator games.

### 3.3.2 How can a computer game be integrated in military training using the WISE platform?

The integration with the WISE platform shall enable the flow of information from the game to the WISE platform and support information flowing from the WISE platform into the game. The following solutions were considered and Solution 2 was chosen because it would be a lot easier and good enough.

#### *3.3.2.1 Modification of network traffic*

Most modern games have multiplayer capabilities. Game A and game B are both in multiplayer mode. The driver could convert multiplayer data from game B to make it look like multiplayer data from game A. To game A, game B would look like another copy of game A. It would be like two instances of game A playing a normal multiplayer game. The same thing could be said about game B. To use this solution, only the network data being sent from each game needs to be modified. Nothing in the game has to be changed. This solution is very difficult to implement, because all network packets must be intercepted, modified and passed on. Aside from the technical difficulty, lag times could increase from the extra overhead and render the system practically useless.

#### *3.3.2.2 Creation of a server-side plug-in and driver*

A server-side plug-in collects the necessary information into a data structure and sends it to the driver that is loaded in WISE. The driver unpacks the information from the data structure and inserts it into WISE. WISE keeps track of all the real-time data according to the information model developed as part of this thesis. Connectivity, which is a part of the WISE platform, lies on top of WISE, can transform and operate on the data and send it back through WISE to the driver. The driver puts the data into the proper data structure and sends it back to the plug-in loaded in the game server. The plug-in extracts the data from the structure and inserts it into the game. Data can be sent both directions and between different types of games.

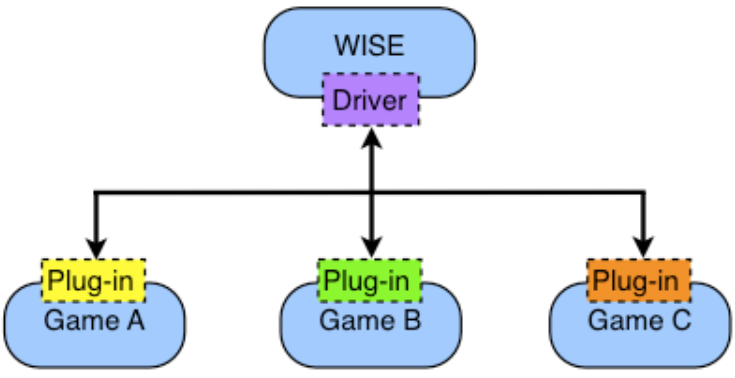


**Figure 12 - Overview of plug-in and driver architecture.**

### 3.3.3 How can a scenario in the MSDL format be imported into a game?

MSDL is a standard that is based on XML and it is used to describe a scenario. It has no concept of time and is only used to initialize a scenario. This part of the thesis is done based on the collaboration thesis (Lundgren & Ullner, 2008). Two possible solutions to import a scenario described using MSDL into computer games were considered. The import through native game script was chosen because implementation would probably be easier and faster. The two solutions do not interfere and could potentially complement each other.

#### *3.3.3.1 Import through plug-in and driver*

A file containing a scenario described using MSDL is opened through the driver. The driver parses the MSDL scenario, and sends commands to each games' plug-in to load the proper map, insert objects into the game, set the properties for each object and to start the simulation. Plug-ins can add, move or remove objects in runtime, while the simulation is running.

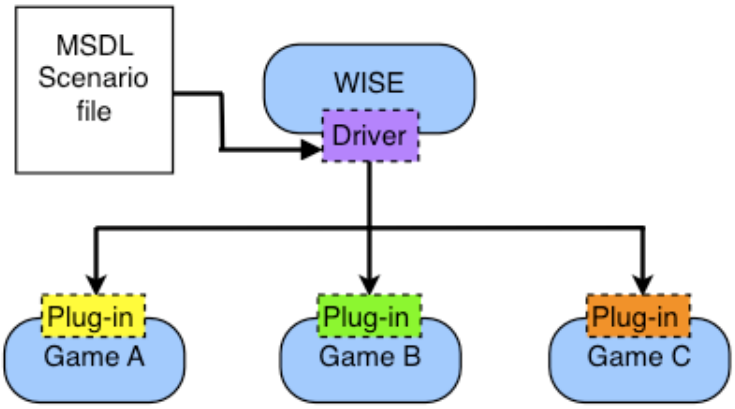


**Figure 13 - Import of scenario through plug-in and driver. A scenario file is loaded by the driver which parses it and orders the plug-ins to setup the scenario through each game's API.**

#### *3.3.3.2 Import through native game script*

A scenario is created in the visual MSDL scenario editor (Lundgren & Ullner, 2008) and exported to native game script files for different games. The driver issues commands to the plug-ins, which in turn make the game server load game scenario scripts. Then clients can run any game that they prefer and that is supported by the system, connect to a corresponding game server and train.

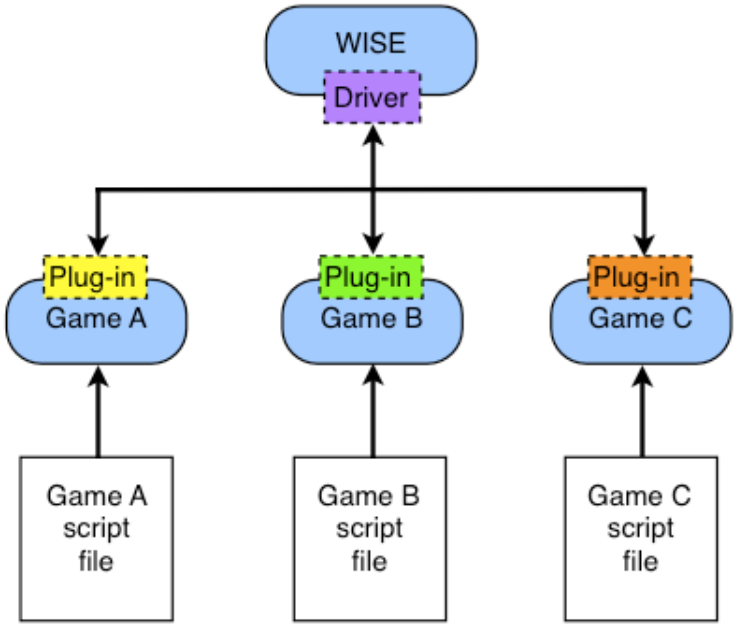


**Figure 14 - Import of scenario through native game scripts.**

## 3.4 Design

The information model was designed by playing several different games, finding similarities and figuring out which information would be required to make different games behave in the same way and look similar. The HL2 Empires plug-in was designed by trial and error, because of ambiguities in the documentation and few working code snippets and examples. The VBS2 plug-in was designed from experiences gained during implementation of the HL2 Empires plug-in and it had less ambiguous documentation.

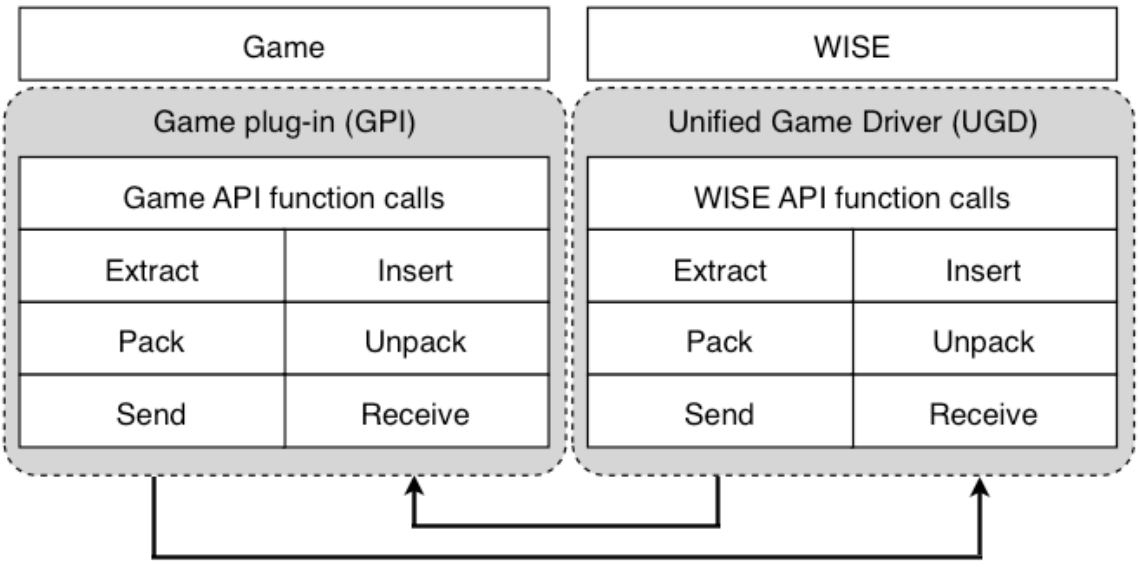


**Figure 15 - Design of game plug-in and unified game driver.**

## 3.5 Implementation

The implementation of the information model is done in STS own proprietary format which is based on XML using their own in-house developed tool, Code. Game plug-in and driver are implemented in C++ using Microsoft Visual Studio. Implementation was done piece by piece, during the design process and tweaked continually until deadline.

## 3.6 Testing

While coding the plug-in and driver, tests are carried out continuously, to ensure proper programming. Each feature and function is tested before work on the next one is commenced. A final test will be carried out with all parts that are finished at deadline.

# 4 Result

This chapter contains detailed descriptions of how the information model, plug-ins and drivers are implemented. No actual application code is included because the implementation contains code provided by STS which is not freely available.

## 4.1 Information model

The purpose of the information model is foremost to make it possible to connect different types of FPS games with each other. This is accomplished by extracting and structuring certain information that is common to most FPS type of games. Examples of common information and actions in FPS type of games are movement, firing, registering hits and communication e.g. receiving/giving orders and objectives.

The information model consists of entities and events. Entities are persistent while events are non-persistent; they are just used to send information and then cease to exist.

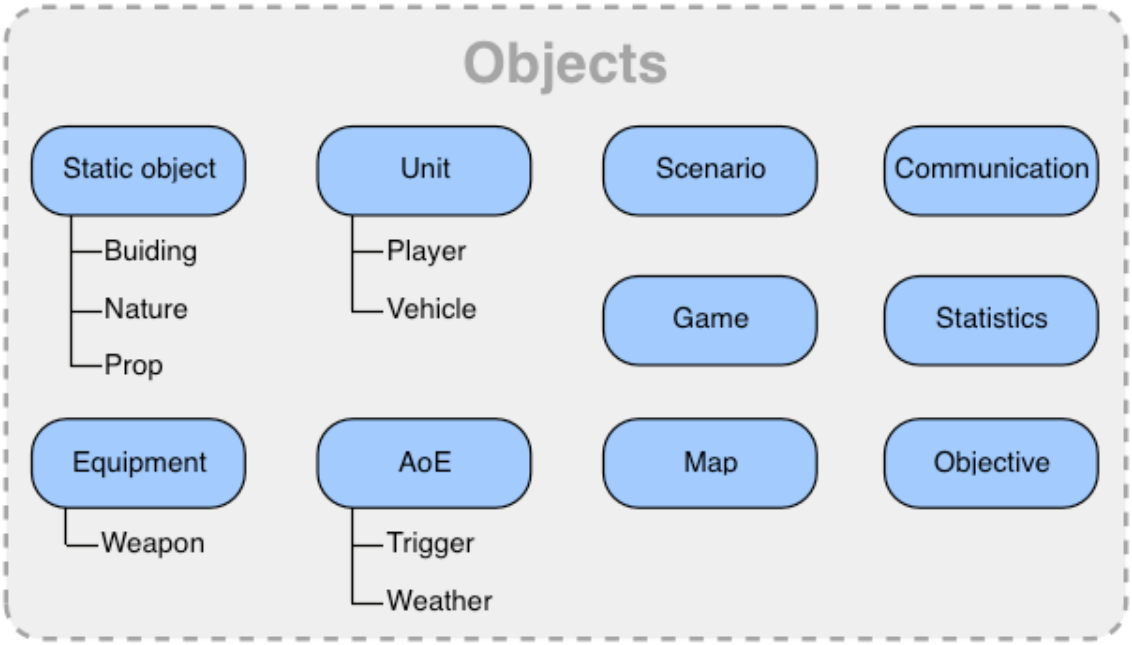


**Figure 16 - Objects in FPS game information model**

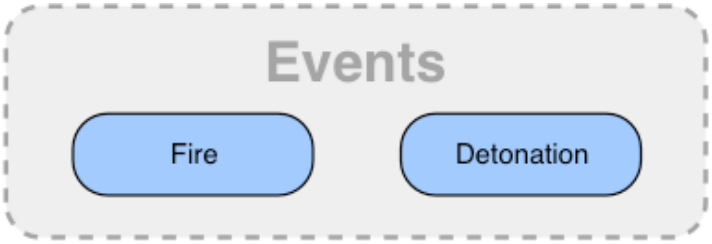


**Figure 17 - Events in FPS game information model**

All italic attributes are required; the rest of the attributes can be given a default value of 0 or null.

### 4.1.1 Object

These attributes are common to all objects.

| Name | Description |
|---|---|
| *ID* | *Unique identifier* |
| Name | Nickname of player, name of building or vehicle |
| *Type* | *Object type; static object, unit or equipment* |

#### *4.1.1.1 Static objects*

Static objects are inanimate objects or nature that cannot be controlled by the player. However, they can be affected by the player, for instance a building can be damaged or destroyed by a player.

| Name | Description |
|---|---|
| *Type* | *Building, nature or prop* |
| *Position* | *Cooridnate of static object* |
| Status | Level of damage, cleared, destroyed, other types can be specified |
| Subtype | Silo (building), debris (prop), tree (nature) etc |

**Subtype**

The types in the table are only examples to illustrate how this attribute could be used. New types can be specified and added if required.

| Name | Description |
|---|---|
| Building | Military, civilian or more specific; silo, warehouse, cottage etc |
| Prop | Crate, barrel, debris |
| Nature | Mountain, boulder, tree, bush |

### 4.1.2 Equipment

The equipment entity is another kind of object. It is not really a static object, because it can be controlled by the player and it is not a unit either because it is not an active entity. Each unit can carry several pieces of equipment, such as armour and several weapons.

| Name | Description |
|---|---|
| *Type* | *Weapon, clothing, communication, armour, misc, other types can be created as needed* |

**Weapon**

Weapons are a type of equipment that can be used by players, to affect other units and static objects.

| Name | Description |
|---|---|
| Name | Name of the weapon, e.g. AK-47, Desert Eagle etc |
| *Type* | *Sniper, Explosive, Rocket launcher, own types can be specified* |
| Ammo | Amount of ammo |
| Inflicted dammage | How much damage each shot inflicts. See WE1 |

**WE1 – Weapon Example 1**

Weapon 1 in game A inflicts 25 amount of damage and weapon 2 in game B inflicts 20 amount of damage. Weapon 1 and 2 are roughly the same across both games, at least of the same type, for instance both are shotguns. The value “Inflicted damage” can be used to make sure that both weapons inflict the same amount of damage. “Inflicted damage” values from weapon 1 and 2 are compared and a ratio is calculated, in this example 25 and 20, 25/20 = 5/4 = 1.25. Every shot from weapon 1 and 2 will make a player in game A lose 20*1,25 (25) health and a player in game B will lose 25*0,80 (20) health. More advanced algorithms could be created by using “Inflicted damage” values together with “Max health” values from the player entity.

### 4.1.3 Unit

The unit entity is the active entity in the game. Unit has two child entities, player and vehicle.

| Name | Description |
|---|---|
| Move direction | Which way unit is headed, relative to its' current position |
| *View direction* | *Which way unit is looking, relative to its' current position* |
| *Position* | *Current position of unit* |
| Max health | Maximum health can be used to calculate damage ratio between different games. See UE1 |
| *Current health* | *Current health of unit* |
| Equipment | List of equipment the unit is carrying, for instance a player can carry several weapons, armour and a vehicle can have weapons attached |

**UE1 - Unit Example 1**

Player 1 in game A shoots player 2 in game B. Player 1 has 400 maximum health, while player 2 has only 100 maximum health. To make sure that they both lose equal amounts of life, the value "Max health" can be used. "Max health" values from player 1 and 2 are compared and a ratio is calculated, in this example 400 and 100, 400/100 = 4. For every health that player 2 loses, player 1 loses 4 health.

#### *4.1.3.1 Player*

The player entity is the only real active entity. It can be controlled by a human player or computer AI.

| Name | Description |
|---|---|
| *Type* | *Military, civilian, terrorist, alien, other types can be specified* |
| Rank | Corporal, major, lieutenant, other ranks can be specified |
| Objective | Players objective(s) |
| Team | Which team player belongs to, company, battalion etc |
| Side | Which side the player belongs to, allied, a country etc |
| *Human* | *Is player controlled by human player or computer AI* |
| Hitzones | The zones on which the player can be hit; arm, head, chest, other hit zones can be specified |
| Statistics | Statistic entity for the player |

#### *4.1.3.2 Vehicle*

Vehicles are not really active entities, but they can contain active entities. Viewed from the outside vehicles will behave just like an active entity.

| Name | Description |
|---|---|
| *Type* | *Military, civilian, terrorist, alien, other types can be specified* |
| Rank | Corporal, major, lieutenant, other ranks can be specified |
| Objective | Players objective(s) |
| Team | Which team player belongs to, company, battalion etc |
| Side | Which side the player belongs to, allied, a country etc |
| *Human* | *Is player controlled by human player or computer AI* |
| Hitzones | The zones on which the player can be hit; arm, head, chest, other hit zones can be specified |
| Statistics | Statistic entity for the player |

### 4.1.4 Map

The map entity keeps track of all static objects and areas of effect that reside in the current map.

| Name | Description |
|---|---|
| Max passengers | Maximum number of passengers that the vehicle can transport |
| *Current passengers* | *List of passenger IDs* |
| *Type* | *Land, Nautical, Air, other types can be specified* |

### 4.1.5 Statistics

The statistics entity keeps track of certain information for each player. By adding together the statistics from each player, statistics for the entire game can be calculated. This object is not required.

| Name | Description |
|---|---|
| Killed | Number of opponents killed by the player |
| Deaths | Number of times player has died |
| Rounds fired | Number of rounds fired by player during the game |
| Target hits | Number of rounds that hit other players |
| Hit | Number of rounds that hit the player |

### 4.1.6 Communication

The communication entity transmits information between units. It can be used to give players orders and objectives and enables units to communicate with each other.

| Name | Description |
|---|---|
| *ID* | *Unique identifier* |
| *Source* | *Reference to the message source* |
| *Destination* | *Reference to the message destination* |
| *Message* | *The actual message, a string* |
| Objective | List of players objectives |

### 4.1.7 Objective

The objective entity is used to tell the player what to do.

| Name | Description |
|---|---|
| Description | Description of the objective, e.g. “Take over an area”, “Eliminate target” |
| AoE | List of AoE that are used to fulfil the objective |
| Priority | The priority of the objective |
| State | Completed, failed, in progress, cancelled etc |

### 4.1.8 Game

The game entity keeps track of information about the game; which map is used, all units that are playing together, overall statistics for the game, keeps track of all ongoing communication and a few other things. The game entity is a runtime entity.

| Name | Description |
|---|---|
| *ID* | *Unique ID of the game* |
| Name | Name of the game |
| Runtime | How long the game has been going on |
| Time of day | Time of day, in the game |
| *Units* | *List of all units who have joined the game* |
| *Map* | *ID of the map used in game, all games and systems should load the same map in their own native format* |
| Statistics | A summary of all players statistics |
| Communication | List of all ongoing communication |
| Scenario | Scenario ID, containing start values for all entities in the game |

### 4.1.9 Scenario

The scenario entity specifies which scenario shall be loaded and points to a scenario file containing start values for all units and static objects in the game.

| Name | Description |
|---|---|
| ID | Unique identifier, see SE2 |
| Name | Name of the scenario |
| Description | Description of the scenario, what it's about, how many players it supports etc |
| File | File name or path to scenario file, for instance a MSDL file. See SE1 |

**SE1 – Scenario Example 1**

When the game is started, the file that the scenario entity points to is parsed and start values for all objects are assigned. A few examples of what can be set up using the scenario entity; objectives for all players and teams, areas of effect on the map like weather and triggers, units can be put in position and assigned to teams and much more.

**SE2 – Scenario Example 2**
Game A starts a game session and a scenario file is chosen, the plug-in sends this information to the driver. WISE registers that a game session is started and what scenario ID it is using. Game B checks what games are available with WISE and discovers that a game session is started. Then game B loads the same scenario file or another native scenario file with the same scenario ID as the scenario in game A. Finally game B joins the game session that game A is in.

### 4.1.10 AoE

The area of effect entity is used to create areas with special properties or effects applied, for instance, weather, minefields and triggers.

| Name | Description |
|---|---|
| *ID* | *Unique identifier* |
| Density | How many mines there are within a square meter, how dense fog, how intense it snows or rains. |
| *Dimensions* | *List of coordinates that make up a polygon where the effect will be applied. Polygons can be three dimensional e.g. volumetric* |
| *Type* | *Trigger or weather, other types can be specified* |
| Description | Description of what area it is, e.g. high radiation, minefield etc |

**Trigger**
Areas of effect of trigger type are used to define an area that will trigger actions. *See AE1*

| Name | Description |
|---|---|
| *Type* | *Action required to engage the trigger, enter, exit, shoot at etc* |
| *Action* | *List of actions to performed, e.g. an explosion, change life, fill ammo etc* |

**AE1 – Area of effect Example 1**
An area of effect of type trigger can be used to do many things, for instance a minefield, an area to restore health and ammo, to receive new orders and objectives, weather can change and much more. When an entity goes into an area of type trigger that is a minefield, an explosion animation and sound can be triggered together with loss of life. The game handles everything, the plug-in and driver transmits information to the WISE, such as the loss of life or player death.

**Weather**
The area of effect entity with type weather is used to specify what kind of weather it shall be.

| Name | Description |
|---|---|
| *Type* | *Snow, Sunny, Fog, Rain etc* |

### 4.1.11 Fire event
An event that is fired every time a shot in the game is fired.

| Name | Description |
|---|---|
| *Source* | *ID of the unit that fired* |
| Destination | Coordinate that is attacked, e.g. units view direction at moment of firing shot |

### 4.1.12 Detonation event
An event that is fired when a unit or static object is hit by something.

| Name | Description |
|---|---|
| *Source* | *ID of the unit that fired the event (unit that was hit)* |
| Hitzone | Where the player was hit, if it isn't a player then this attribute has no value |
| Damage | Amount of health unit loses |

## 4.2 E/R for FPS game information model

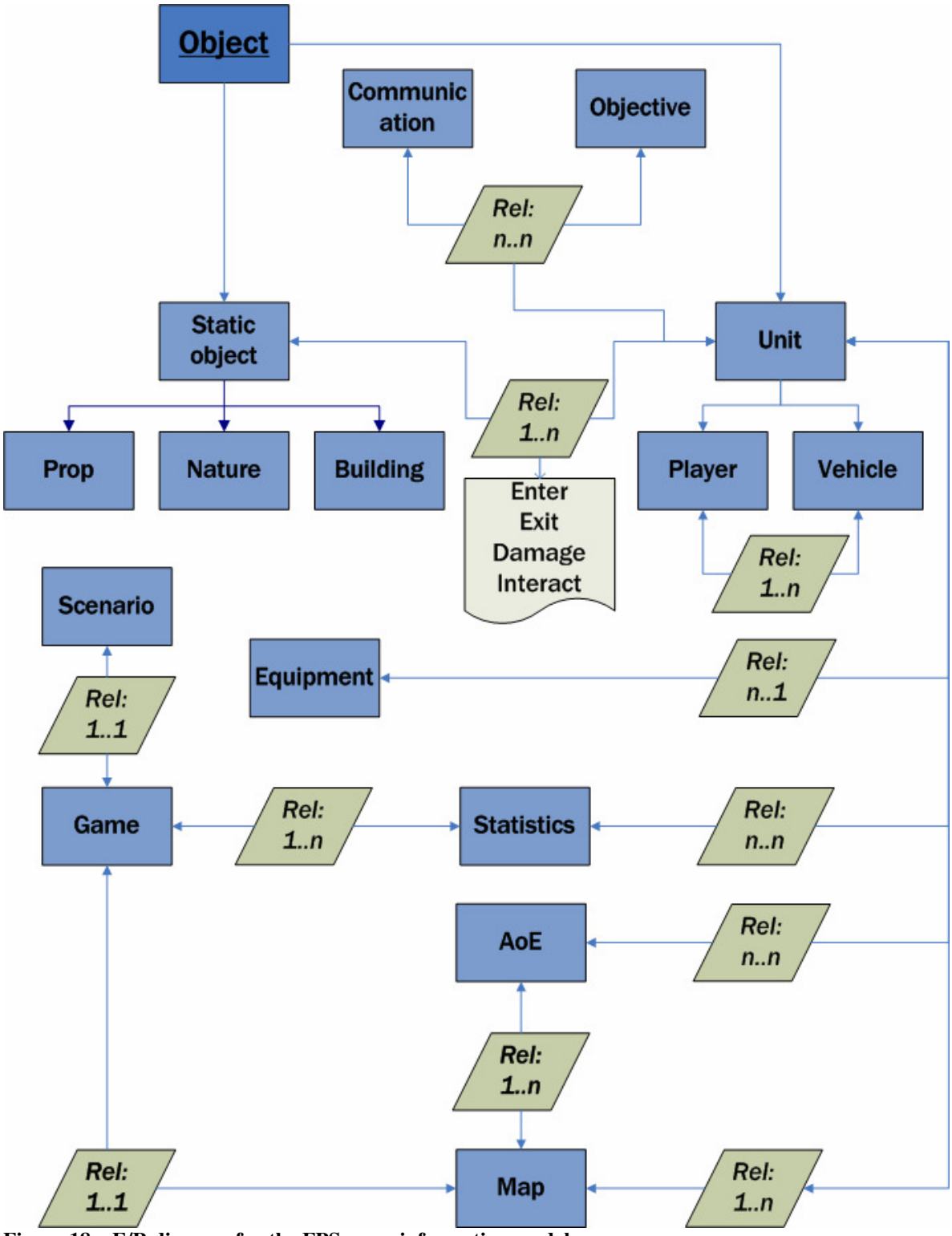


**Figure 18 – E/R diagram for the FPS game information model.**

### 4.3 Implementation of plug-ins and driver

The game plug-in is a software extension in the form of a dynamic link library file (DLL). It uses the game framework to access information about objects in the computer game and is written using each game's API.
This thesis incorporates the implementation of plug-ins for HL2 with the Empires modification and VBS2. Both plug-ins follow the same concept [Figure 15 - Design of game plug-in and unified game driver.] but are implemented differently because of the difference between the games' APIs and the features that they support. The driver does the same things as the plug-in, in terms of communication; the difference is that the driver has to handle communication between several games.

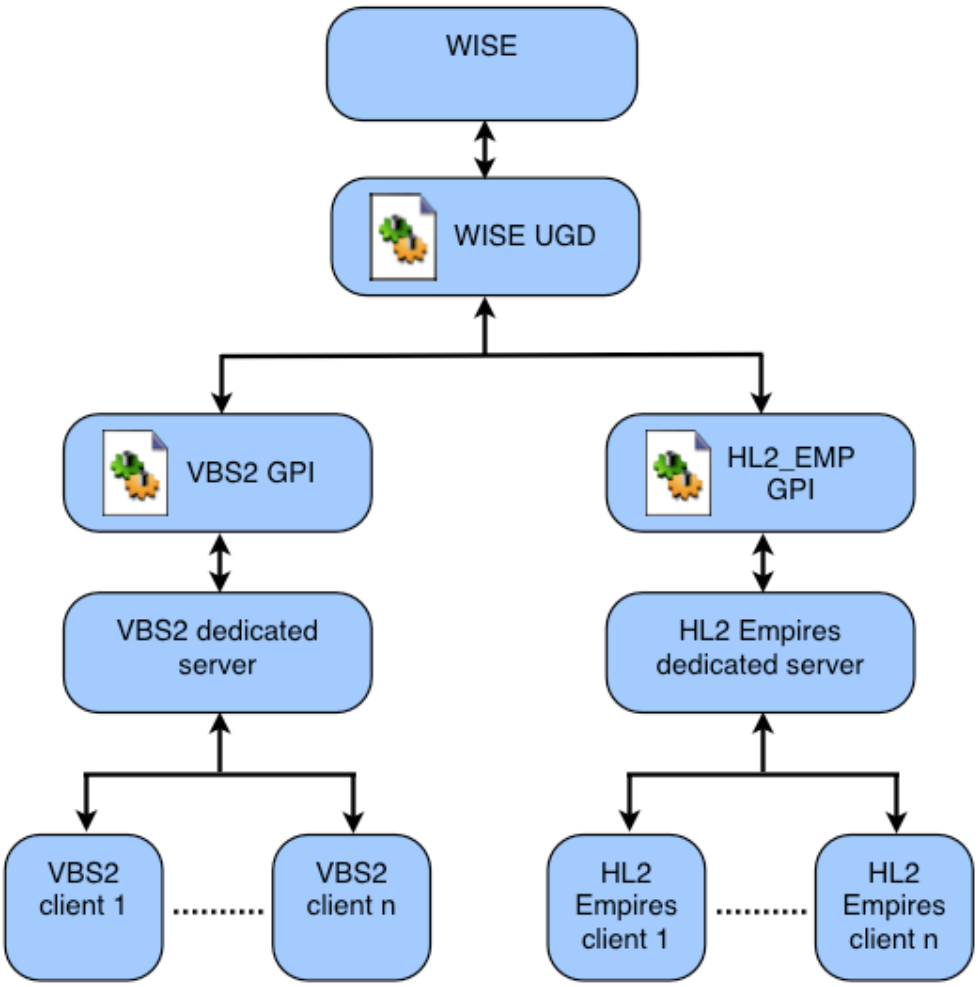


**Figure 19 - Overview of implemented architecture. UGD (Unified Game Driver), GPI (Game plug-in).**

The WISE UGD is the driver which all the game plug-ins uses to communicate with WISE. Each plug-in load with a game server and then clients connect to their respective server.

### 4.3.1 Half-Life 2 Empires plug-in

The Source engine SDK contains C++ classes and functions that can be accessed directly by a plug-in. There is a limited amount of functions and classes included in the SDK, which are accessible by plug-ins, as the diagram to the left shows in figure 20. However, it is possible to access more functions through the use of hacks created by the modification communities. The hacks scan the main memory for signatures of virtual functions in the game, effectively exposing more of the games functionality, as is shown in the diagram to the right in figure 20.

The implementation of the HL2 Empires plug-in requires Source Metamod (Anon, 2008, Metamod:Source for Half-Life 2), which provides access to functions needed to retrieve a lot of data from the game, which is not accessible otherwise.

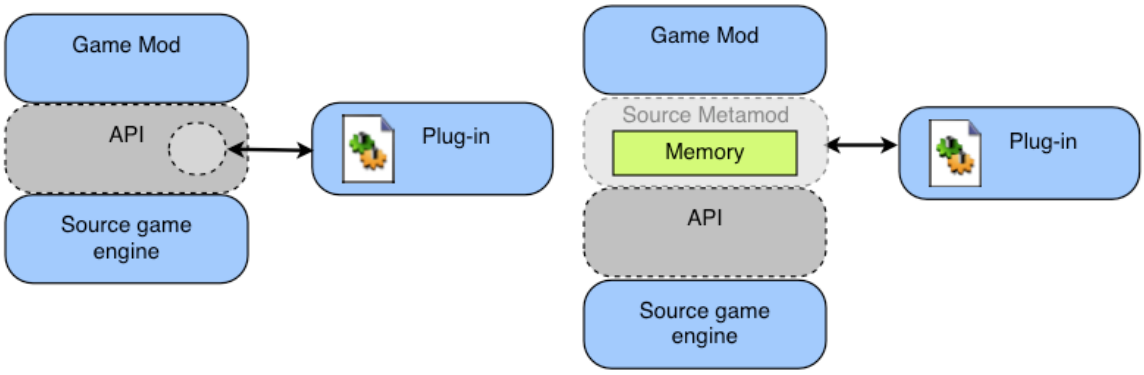


**Figure 20 - Overview of HL2 and Source Metamod architectures.**

When the plug-in loads, certain functions in the Source API are hooked using Source Metamod API and the plug-in subscribes to the desired game events, such as player death, hit, connect etc, represented by the INIT box in figure 21. When a client connects to the server the OnGameFrame-function is entered. The OnGameFrame-function is run on every game frame of the server, which usually runs at 33 frames per second. A timer makes sure that information about entities; such as position and current health for all players, is collected and sent once every second. All data is sent through a socket interface to a simple socket server, using TCP. When events are fired by the game engine, the FireEvent-function is called, the type of event is determined and related data is sent instantly. There is also a utility-class serving as a placeholder which is supposed to contain helper functions.

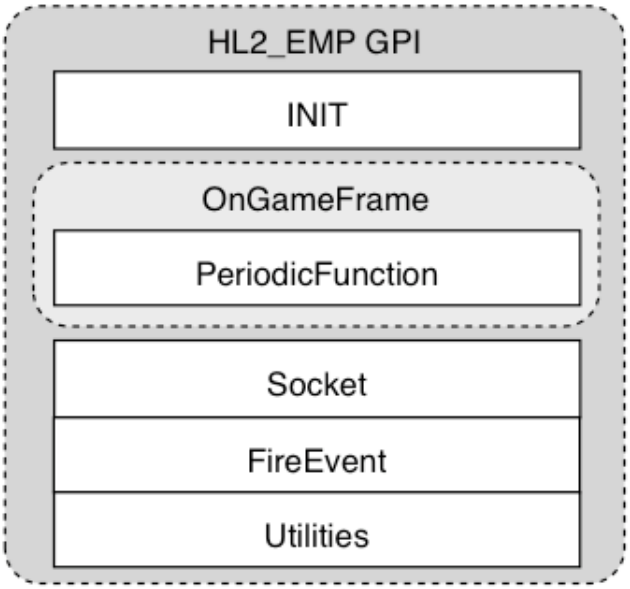

**Figure 21 - Diagram of the HL2 Empires plug-in implementation.**

#### *4.3.1.1 Future improvements*

There is a lot to improve on the plug-in because development was halted and focus was shifted to the VBS2 plug-in. A very simple socket server was created to test the plug-in. The socket part of the plug-in is designed to work with the simple socket server. More advanced socket classes, which are threaded, were provided by STS and turned into a static library. This new library can be used in all future game plug-ins. The socket part of the HL2 Empires plug-in has to be rewritten to take advantage of the new socket library. FireEvents and PeriodicFunction need to be rewritten and expanded to comply with the new unified game information model [Chapter 5.1]. Functions to inject data into the game have not been implemented at all; they are required to use other games together with HL2.

### 4.3.2 VBS2 plug-in

There was no SDK available for VBS2 at the time of writing this thesis, although there was extensive scripting support. Entire scenarios can be scripted, objects in the game can be manipulated in runtime and information can be retrieved by executing script commands. Late in the process of writing this thesis Bohemia released support for VBS2 plug-ins that could execute script commands which opened up possibilities to get information in and out of the game. STS requested to put the HL2 integration on hold, document it for future use and begin work on integrating VBS2 instead.

When the plug-in loads, the game engine passes a pointer to a function, which can execute script commands in the game, represented by the "DLL init function" box in figure 22. The game calls a function, OnSimulationStep, in the plug-in on every game frame. OnSimulationStep contains a timer that makes sure that information about units is sent every second. It adds event handlers for all units, according to figure 24. When events are fired the PluginFunction is called. Game events and entity state data is compliant with the unified game information model [Chapter 5.1]. However, not all data in the information model is retrieved.

The unit positions sent to the driver are converted to a geodetic coordinate system containing longitude, latitude and altitude to be able to map the game coordinates to any place on earth.

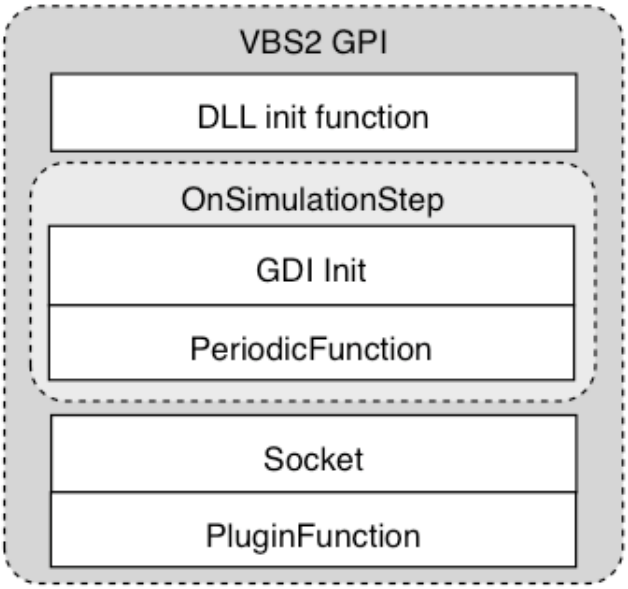


**Figure 22 - Diagram the VBS2 plug-in implementation.**

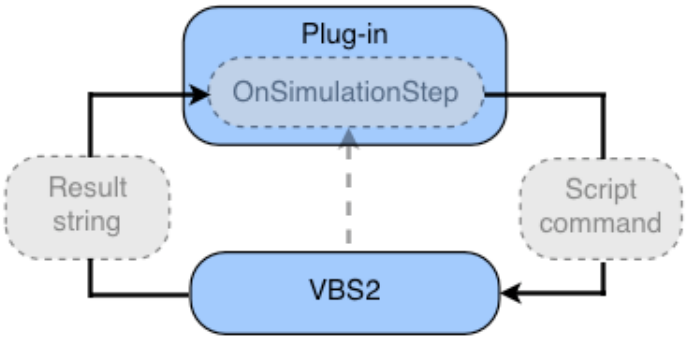


**Figure 23 - Overview of the VBS2 plug-in architecture. The dashed arrow is a pointer passed to the plug-in.**

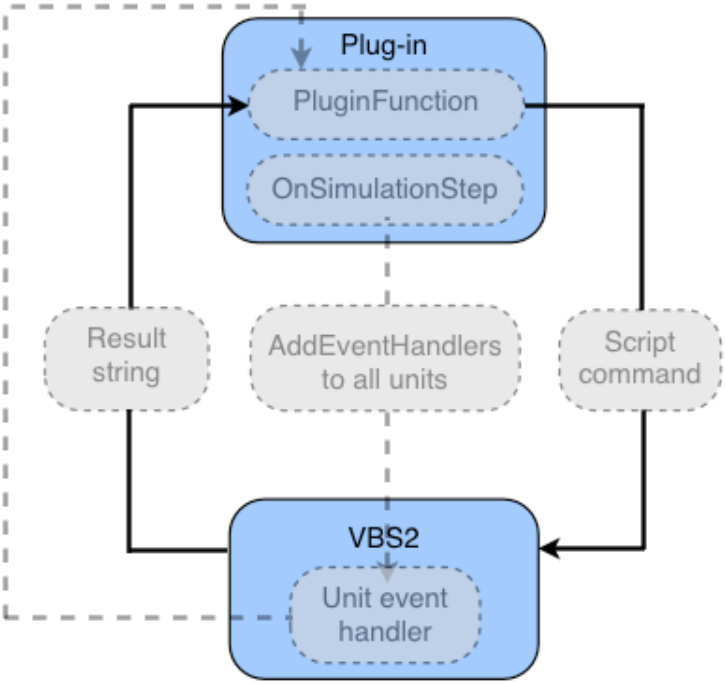


**Figure 24 - Overview of VBS2 plug-in event-handlers.**

#### *4.3.2.1 Future improvements*

Extracting more data according to the information model [Chapter 5.1], injection of information into the game and import of scenarios described using MSDL.

### 4.3.3 Unified game driver

The driver is designed to work with all types of games that make use of the unified game information model. The WISE platform is only required to load one instance of the unified driver to communicate with all connected computer games. The driver receives information from several games through the OnReceiveSocketData-function. It creates and updates objects in the unified game information model database. Updated data from the information model database are sent to the connected games by the SendInfoToGame-function.

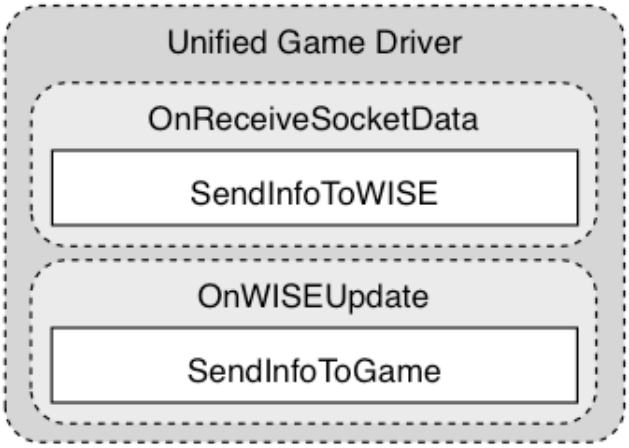

**Figure 25 - Diagram of Unified Game Driver implementation.**

The data according to the unified game information model is split up into several structures and sent to the driver. For example, the structure containing information about a unit looks like this:

```
struct EntityState
{
        HeaderStruct header;
        wchar_t id[50];
        wchar_t name[80];
        float current_health;
        Coordinate position;
        ...
};
```

**Example 1 - EntityState structure in the unified game information model, implemented in C++.**

#### *4.3.3.1 Future improvements*

The communication protocol UDP allows each game to belong to a multicast group. Using this technique, the driver can send information to each game individually, multiple selected groups or to all connected games using broadcast, however it is not used.

## 4.4 Limitations of the chosen game platforms

The chosen platforms have certain limitations which affect development and which applications the platforms are suited for.

### 4.4.1 HL2 Empires

The game chosen in at the beginning of this thesis was Half-Life 2 from Valve. Valve has a platform called Steam which is used to install, start, authenticate and update its games. Steam provides some interesting possibilities and a few major drawbacks. It makes it easy to organize games,

modifications, developer tools and to keep everything updated. There is one huge downside, if Steam fails to start, then it is not possible to play the games that depend on it. Steam also needs an Internet connection to authenticate the game licenses every time a game is started, thus making it impossible to play without an Internet connection. It is possible to reverse engineer Steam and circumvent the protection, but it is illegal and unpractical. A possible solution could be to make a deal with Valve, they could make a special military plug-in, license or Steam version that would not require authentication or an Internet connection to run. There is another limitation with the Source engine; Half-Life 2 Multiplayer, Counter-Strike: Source, Day of Defeat and Team Fortress all have different player classes and slightly different code-bases. Therefore it is difficult to write a plug-in that can work with all the Source based games without writing custom code for each game.

### 4.4.2 VBS2

The only way to communicate with VBS2 via plug-ins is using scripting commands; therefore the limitation is the extent of the scripting language. Results returned from the script commands return strings; this is a great inconvenience because the strings have to be parsed to extract the information requested.

# 5 Results based on the collaboration thesis

Parts of this project are made using work from a collaboration thesis (Lundgren & Ullner, 2008) also written at STS. The collaboration thesis goals are to create an information model for strategy type of games and a scenario editor that can export scenarios to MSDL format. The collaboration will target importation and exploring the scripting languages of VBS2 and HL2. The information models from each project shall merge into one generic information model that works with both FPS and strategy type of games.

## 5.1 Combined information model

This is the combined information model that is targeted at FPS and strategy type of games.

### 5.1.1 Entities

The FPS model consists of the following entities:

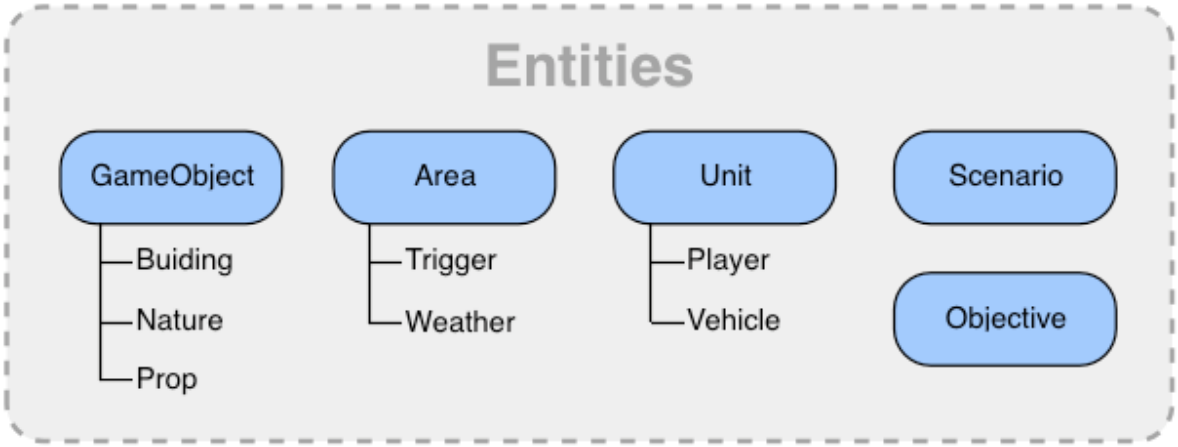


**Figure 26 - Entities in unified game information model.**

#### *5.1.1.1 Units*

The unit entity is the active entity in the game. There are two types of units, player and vehicle.

| Name | Description |
|---|---|
| *ID* | *Uniqe identifier* |
| *Object type* | *Player or vehicle* |
| Name | Nickname of unit |
| *Position* | *The position of the unit* |
| View direction | Direction the unit is facing |
| Speed | Current speed of unit. Measured in m/s |
| *Current health* | *Current health of unit* |
| Max health | Maxumum health of unit |
| *Bot* | *Human or computer AI* |
| Commander | The commanding unit |
| Subordinates | Unique identifiers of units that the unit can command |
| Weapon | List of weapons the unit is carrying. First weapon in list is the current weapon |
| Weapon range | Range of the current weapon, measured in meters |
| Ammo | Current ammo left in current weapon |
| Equipment | List of equipment the unit is carrying |
| Objectives | List of objectives the unit is supposed to pursue |
| Time | Time experienced by unit |
| Transport vehicle | Which team or group the unit belongs to |
| Rank | Rank of unit, e.g corpral, major, lieutnant etc |
| Team | Which team or group the unit is in |
| State | What state the unit is in (jumping, crawling etc). Depends on the object type |
| Hitzone | List of hitzones where unit has been hit (depends on the object type) |
| Hits | Number of times the unit has been hit |
| Kills | Number of units killed by the unit |
| Rounds fired | Number of rounds fired |
| Deaths | How many times the unit has died (if it is possible to respawn) |
| Target hits | How many times the unit has hit targets |

### 5.1.2 Area

The area entity is used to create areas with special properties or effects applied, for instance, weather, minefields and triggers.

| Name | Description |
|---|---|
| *ID* | *Uniqe identifier* |
| *Object type* | *Weather or trigger* |
| *Position* | *Position of area center* |
| *Dimensions* | *List of coordinates that make up the 3D-area* |
| Density | The density of the area, depends on object type, for instance how many mines per square meter in a minefield or how intense it is raining etc |
| *Type* | *Type of weather (snow, rain) or trigger (enter, exit). Depends on object type.* |
| Description | Description of what the area does, e.g. “Heavy rain”, “Calls airstrike upon unit entry” |
| Action | List of actions to be performed, for examsple an explosion, change life, fill ammo etc, required if object type is trigger |

### 5.1.3 Events

Events are used to transmit information that does not have to be persistent.

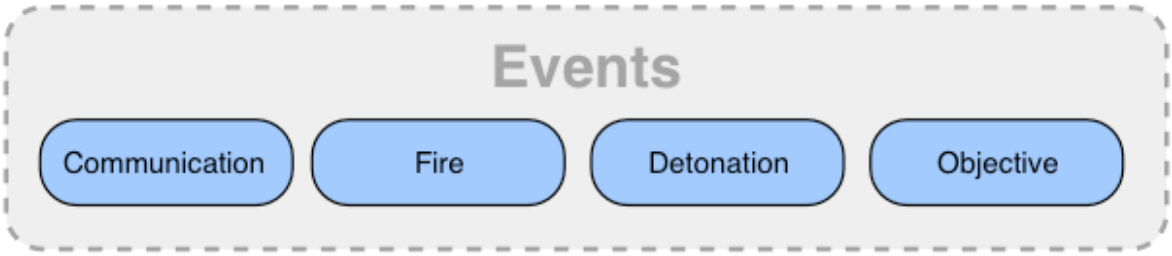


**Figure 27 - Events in unified game information model.**

#### *5.1.3.1 Fire*

An event that is fired every time a shot in the game is fired.

| Name | Description |
|---|---|
| Position | Position of the firing unit |
| Destination | Coordinate of source units crosshair when shot fired |
| *Source* | *Unique identifier of the unit firing* |

### *5.1.3.2 Detonation*

Event that is fired when a unit or static object is hit by something.

| Name | Description |
|---|---|
| Position | Position of impact, can be used to make craters and decals appear in the same position across different games. |
| *Destination* | *The unique identifiers of the units that should receive the message* |
| *Source* | *Unique identifier of the initiating unit* |
| Hitzone | Where unit was hit, depends on the “object type” of unit |
| Damage | Amount of health that unit loses |

### *5.1.3.3 Communication*

The communication event transmits information between units and can be used to give players orders.

| Name | Description |
|---|---|
| *ID* | *Uniqe identifier* |
| *Destination* | *The unique identifiers of the units that should receive the message* |
| *Source* | *Unique identifier of the initiating unit* |
| *Message* | *Actual text message* |

### *5.1.3.4 Objective*

The objective event is used to tell the player what to do and specify triggers that determine when the objective has been completed. The event can trigger the creation of an object of type objective and copy the information from the objective event to the objective object.

| Name | Description |
|---|---|
| *ID* | *Unique identifier* |
| *Source* | *Unique identifier of the initiating unit* |
| *Recipients* | *The unique identifiers of the receiving units* |
| Priority | Priority of the objecive |
| State | State of which the object is in, e.g. cancelled, in progress or completed |
| Timeout | How long the objective is valid in seconds. 0 means permanent |
| *Description* | *Description of the objective* |

## 5.2 E/R for the Unified game information model

This chapter explains the relationships between the objects in the unified game information model.

The dynamic object block contains units and vehicles. The static objects block contains GameObjects which can be nature elements, props and buildings.

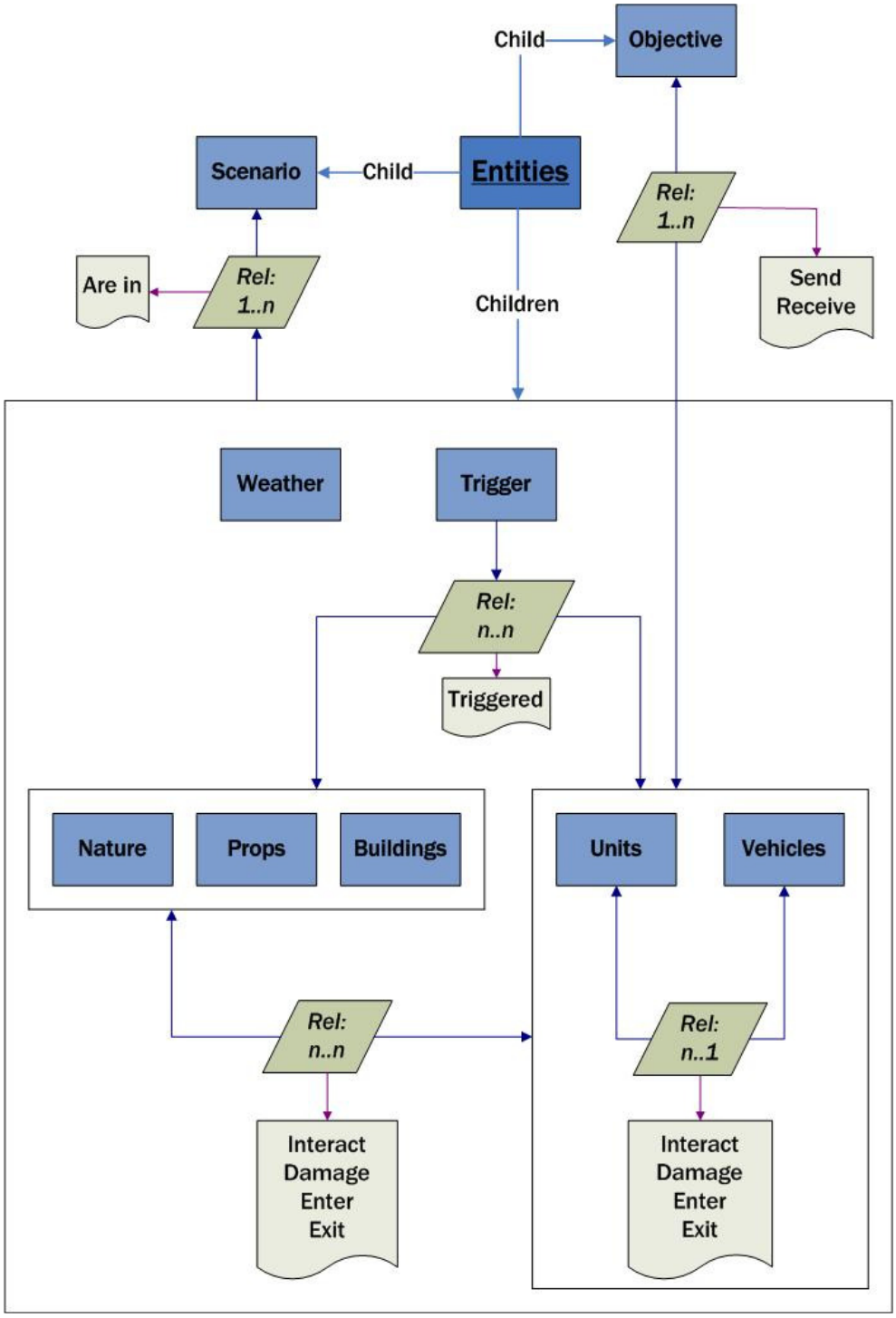


**Figure 28 – E/R diagram for the Unified game information model**

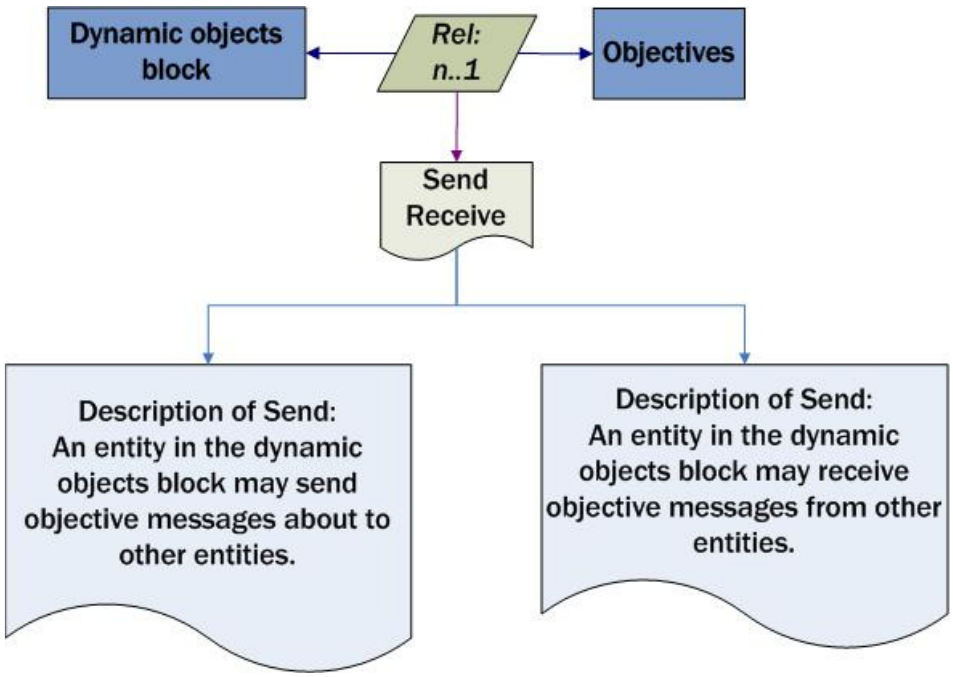


**Figure 29 - Relation between dynamic objects block and objectives.**

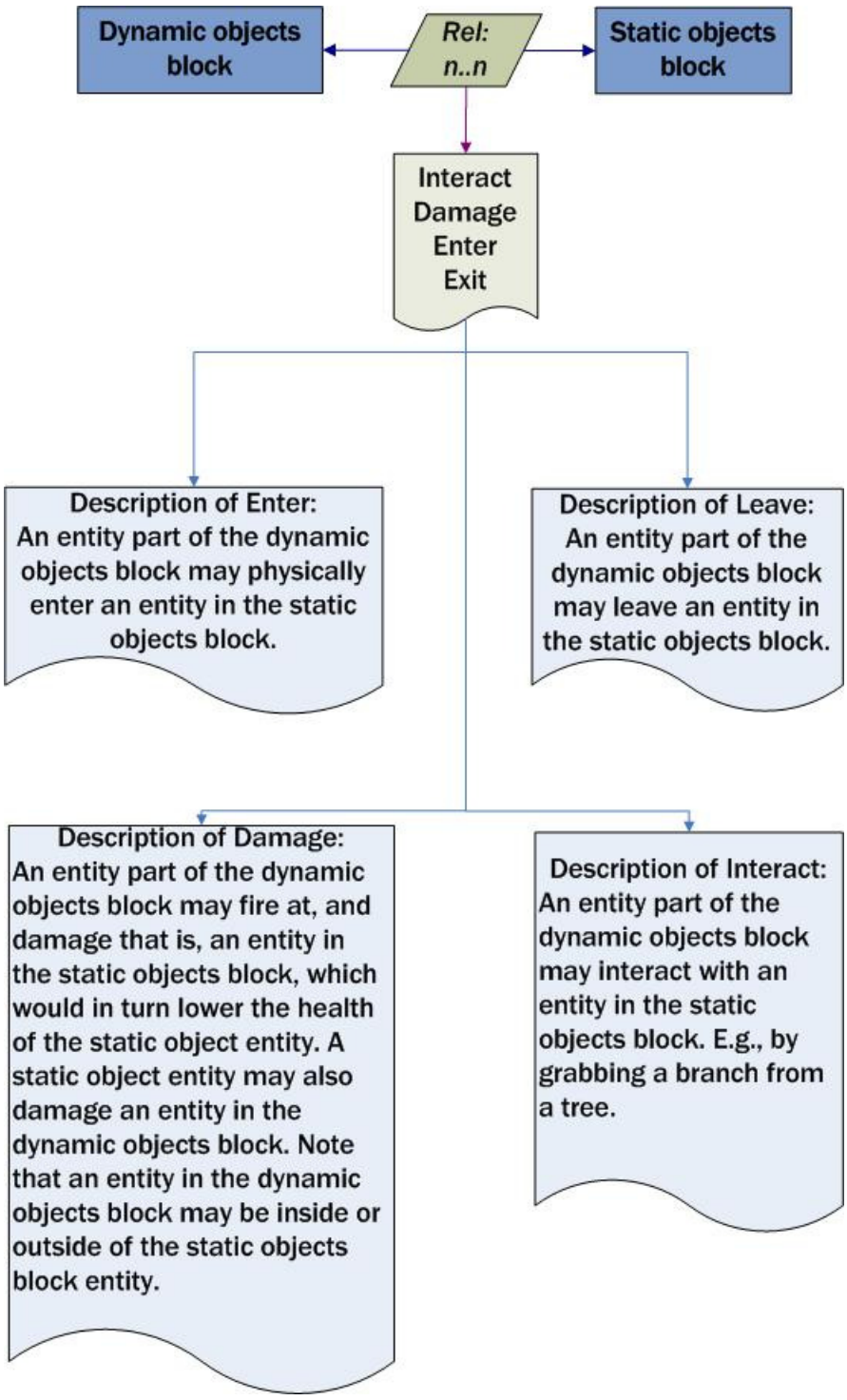


**Figure 30 – Relation between dynamic objects block and static objects block.**

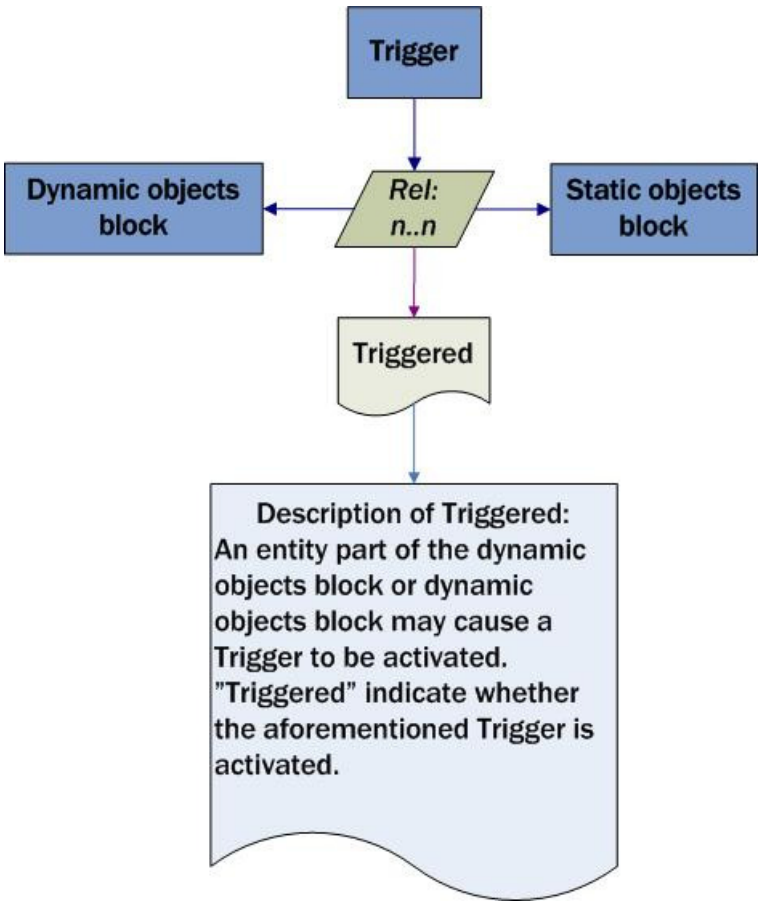


**Figure 31 – Relations between the dynamic objects block, the static objects block and triggers.**

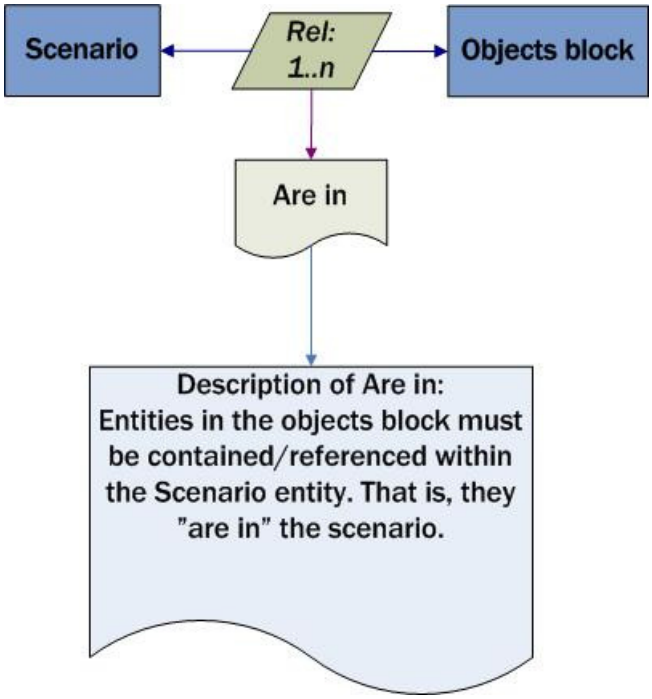


**Figure 32 - Relation between scenario and the objects block. The object block is the big block in figure 28.**

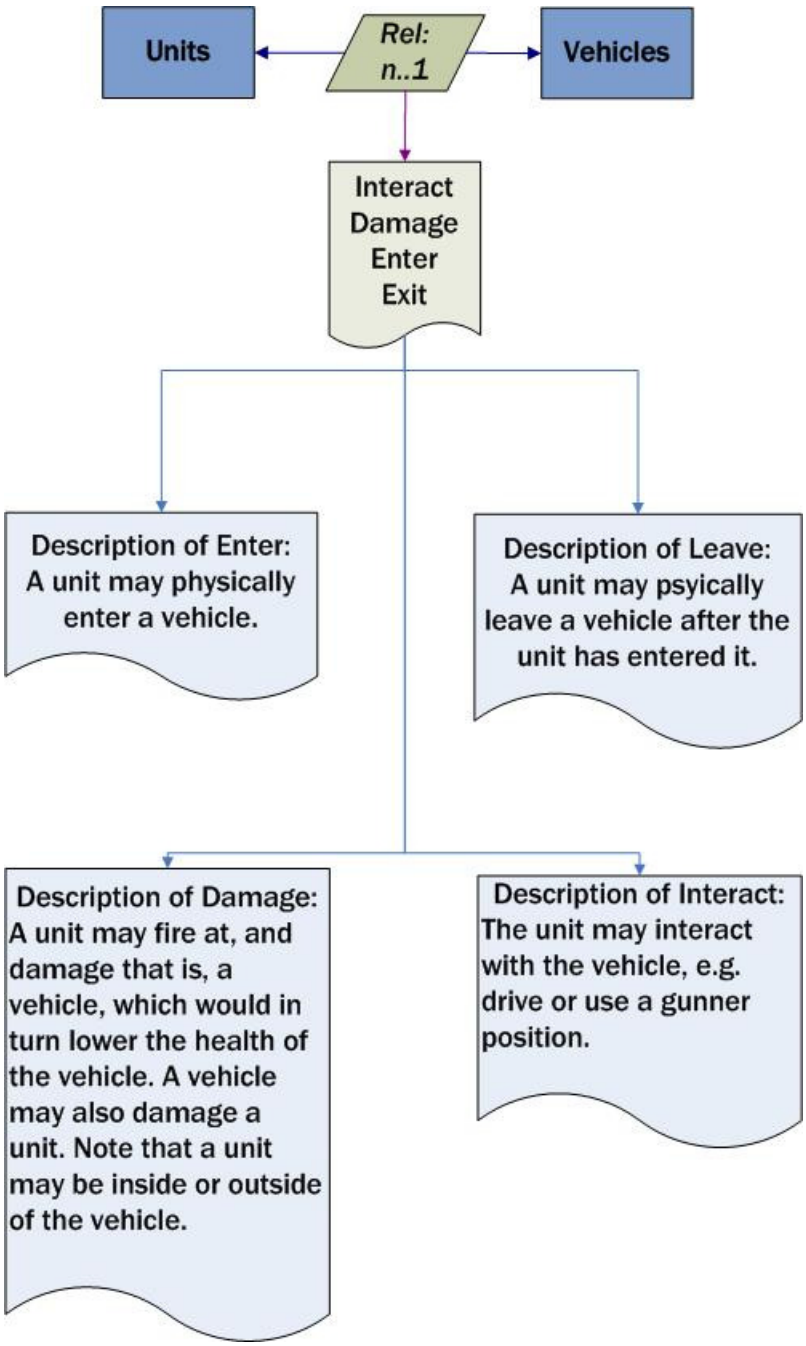


**Figure 33 - Relation between units and vehicles.**

## 5.3 Exploration of scripting for games

The exploration and documentation scripting languages in VBS2 and HL2 are meant to aid the collaboration thesis (Lundgren & Ullner, 2008) in implementing exportation to each scripting language in their scenario editor.

### 5.3.1 VBS2 scripting

VBS2 has built in support for scripting and the scripting language is extensive. With VBS2 1.19, plug-ins can execute any command supported by the scripting language in run-time to extract or inject information.

Script command:

```
getPos (allunits select 0)
```

**Example 2 - Get the position of a unit using VBS2 script.**

Script command:

```
(allunits select 0) addEventHandler [“KILLED”,
hint{format [“%1 killed by %2”, _this select 0, _this
select 1]}]
```

**Example 3 - Add a killed-event to a unit and display a message when the event is triggered using VBS2 script.**

### 5.3.2 Half-life 2 scripting

There is currently no official scripting support for HL2. However, there are 3rd party plug-ins/modifications which enable bidirectional communication with the game through the plug-in/modifications specific scripting language.

#### *5.3.2.1 SourceMod scripting*

SourceMod is a Half-Life 2 modification enabling creation of plug-ins for server modification and administration. Plug-ins are scripted in the SourcePawn language, enable scripting actions on game servers and access a lot of the restricted functionality in the Source engine.

```
public OnPluginStart()
{
   HookEvent("player_death", Event_PlayerDeath)
}

public Event_PlayerDeath(Handle:event, const String:name[],
bool:dontBroadcast)
{
   new victim_id = GetEventInt(event, "userid")
   new attacker_id = GetEventInt(event, "attacker")

   new victim = GetClientOfUserId(victim_id)
   new attacker = GetClientOfUserId(attacker_id)

   /* Do something with the event… */
}
```

**Example 4 - Hooking the player_death-event using SourceMod scripting.**

### *5.3.2.2 EventScripts*

EventScripts provide an easy-to-use scripting interface to code add-ons that change gameplay or add new functionality to gameservers.

```
block load
{
  es_msg My hello script has been loaded.
}

block unload
{
  es_msg My hello script has been unloaded.
}

event player_spawn
{
  es_msg Hello World! event_var(es_username) has spawned!
}
```

**Example 5 - Hooking the player_spawn-event using EventScripts.**

# 6 Conclusions

The questions this thesis is meant to answer where:

- How can a computer game be integrated in military training using the WISE platform?
- What information does FPS and strategy games have in common?
- How can a scenario script in the MSDL format be imported into a computer game?

A game can be integrated in military training using the WISE platform by developing a server-side game plug-in for the game and a driver for WISE. Together the plug-in and driver provide a communication path to the game. The information most FPS games have in common is fire and detonation types of events and a periodic update of entity data such as position and status.

There are several answers for the third question; one solution is to translate the MSDL file to native game script and load it either through the game or through the driver. All three questions are answered in this thesis and the first two questions are implemented at a proof-of-concept level.

The goals of this thesis were to make an implementation of the answers, which means that two out of three goals have been fulfilled. The third goal, which was to create a way to import MSDL scripts was not implemented because the implementation of the plug-in and driver took longer than expected, the game that was going to be integrated was changed at half way to deadline and it was not one of STS' top priorities. The acceptance requirements were fulfilled and STS are content with the product that was delivered.

The algorithm used during this thesis consists of three phases corresponding to the assignments requested by STS. The FPS information model was completed during phase 1 as well as the first draft of the Unified Game Information model. Phase 1 was extended a few weeks because of restricted Internet access. Some work could have been done from home, for example HL2 could have been tested and examined at home.

Phase 2 consisted of developing a WISE-driver and a plug-in for HL2 and lasted until deadline, a few weeks before deadline the focus of integration was shifted to from HL2 to VBS2. Driver development started three weeks before deadline and finished a week after deadline. While waiting for the driver development course that was promised by STS, focus was on plug-in development.

Phase 3 was supposed to focus on MSDL scenario import and exploration of game scripting languages, but due to the changes in phase 2 and a limited amount of time, only the exploration of game scripting was carried out. The exploration of game scripting was carried out in parallel with phase 2 and essential to access data in HL2 and the only way to create plug-ins for VBS2.

Risk analysis was performed and re-evaluated at checkpoint 1 and 2, it was supposed to be done at every checkpoint but was forgotten due to heavy workload. During phase 1 the greatest risks were; too large scope of project because the scope of the project was uncertain, bad time planning because the scope of the project could prove to be larger than expected. At checkpoint 1 the risk analysis was updated and the greatest risks were; lack of tools, new requirements and demands from STS late in the project. There was a lack of tools and STS did come with new requirements. While waiting for tools other parts of the project, not dependant on the tools, were completed for instance authoring this thesis. New demands from STS were prioritized and some were implemented and some discarded.

## 6.1 Comments

This thesis provided many different experiences. We got to use and expand our knowledge in technical fields such as C++ programming, network technology and system design. We also got some business insights of the military industry and what STS' business is.

There were several occurrences which caused delays and some frustration on our part. The game to be integrated was Empires. All Source engine based games need a program called Steam which is used to manage games. Steam checks game licenses over the Internet using certain ports, which were blocked by firewalls outside the reach of STS. Saabs IT department that handles all computer resources refused to open up the ports. Therefore the game to be integrated could not be started, plug-in and driver development was delayed. The computers connected to STS' network had restrictions on the size of files that could be downloaded, refusing to download the Empires modification and other large files that were needed. It was not possible to share a folder with a non STS' configured computer connected to the protected network; there was no DVD-burner and USB memory sticks were disabled on the computer connected to STS' network. It was not possible to move large files through the network or through physical media.

STS provided unrestricted Internet access using a 3G modem and service; however, the connection was slow and unstable. Finally a 3G router was provided which provided satisfactory access to the Internet. Work on the game

plug-in progressed slowly because examples and documentation found on the Internet was ambiguous and often did not work. Later we found out that games based on the Source engine do not use the same code base for entities, which explains the ambiguous documentation found on the Internet.

A one and a half month from deadline the game to be integrated was changed by STS to VBS2 by Bohemia Interactive, because a new version of the game which supports plug-ins was released. Luckily, VBS2 was easier to develop for because there is only one codebase and the experience gained developing the Empires plug-in allowed us to create a plug-in very quickly.

Driver development did not start until a few weeks from deadline because we did not receive the driver development course that was promised. Our supervisors at STS had a high workload and could not provide us with the course earlier, although it worked out because of the change of game to be integrated. There was no time left for developing support for importation of scenarios described using MSDL.

All these experiences where good because this is how things work in reality, there are often delays, requirement changes and other things that do not happen optimally. Because we got to experience this type of events we will be better equipped to handle them when they happen in the future.

We came up with a few tips for future theses:

- Divide the work into as many independent parts as possible, that way if a part cannot be carried out until a certain condition is met; it is possible to work on other parts in the meantime.
- In school you learn the how things should be done. However, oftentimes it is necessary to temporarily deviate from the proper ways and use less optimal solutions to have a working product on time.

## 6.2 Possible improvements

Some improvement suggestions to STS are to create an open laboratory environment without firewall or to use a configurable firewall and clean Windows XP installations with full administrative privileges for future theses.

The work methods used in this thesis could be improved in several ways:

- Requesting the supervisor to check the progress more often.
- Dividing the work labour differently between the authors, allowing an equal part of programming and authoring the thesis.
- Follow time schedule more strictly, working strictly from 09.00 to 17.00 o'clock.

### 6.2.1 Suggestions for future theses

This chapter contains suggestions for future theses which can be based upon this thesis.

- Extend the functionality of the VBS2 and HL2 Empires plug-ins and drivers to make it possible to play together using totally different game engines. The plug-in and drivers could be refined from proof-of-concept to production quality by implementing the entire information model and testing driver and plug-in thoroughly.
- Extend the information model further to support more features and games for instance flight simulators.
- Develop new information models for civilian purposes for instance police and law enforcement, rescue missions involving fire-, ambulance- and other rescue personnel.

# 7 References

Jakob Blomberg (2008) [Presentation about Serious Gaming] (Personal communication)

Adam Lundgren & Fredrik Ullner, 2008. Lessons learned from implementing a MSDL Scenario Editor – A tool for the serious gaming community. LTH School of Engineering.

Anon, 2008, Virtual Battle Space. [Online].
http://virtualbattlespace.vbs2.com/
[2008-05-27]

Anon, 2008, The Orange Box. [Online].
http://orange.half-life2.com/
[2008-05-27]

Anon, 2008, Empiresmod.com. [Online].
http://www.empiresmod.com/
[2008-05-27]

Anon, 2008, CALL OF DUTY. [Online].
http://www.callofduty.com/
[2008-05-27]

Anon, 2006, EA ::Battlefield 2. [Online].
http://www.battlefield.ea.com/battlefield/bf2/
[2008-05-27]

Anon, 2008, Metamod:Source for Half-Life 2. [Online].
http://www.sourcemm.net/
[2008-05-27]

Anon, 2008, Virtual Battle Space. [Online].
http://virtualbattlespace.vbs2.com/index.php?option=com_content&task=view&id=82&Itemid=79
[2008-05-27]

Anon, 2008, GLOBAL MAPPER. [Online].
http://www.globalmapper.com/
[2008-05-27]

Adam Lundgren & Saša Perak & Robert Michalski & Fredrik Ullner, 2007. Final Report – Virtual Battlespace 2. November 2007 ed. [pdf] Lund University, LTH School of Engineering.

Anon, 2008, SDK Docs – Valve Developer Community. [Online].
http://developer.valvesoftware.com/wiki/SDK_Docs
[2008-05-27]

Anon, 2008, Category: Third Party Tools – Valve Developer Community. [Online].
http://developer.valvesoftware.com/wiki/Category:Third_Party_Tools
[2008-05-27]

Anon, 2008, Maya to HL2 integration – Facepuch Studios. [Online].
http://forums.facepunchstudios.com/showthread.php?t=477891
[2008-05-27]

Anon, 2008, SourceMod – Half-Life 2 Scripting. [Online].
http://www.sourcemod.net/
[2008-05-27]

Mattie Casper, 2007, Mattie{info}. [Online].
http://mattie.info/cs/
[2008-05-27]

Institute of Electrical and Electronics Engineers, 1998, IEEE 1278.1A-1998 - Standard for Distributed Interactive Simulation - Application protocols, [pdf] Institute of Electrical and Electronics Engineers.

Institute of Electrical and Electronics Engineers, 2000, IEEE 1516-2000 - Standard for Modeling and Simulation High Level Architecture - Framework and Rules, [pdf] Institute of Electrical and Electronics Engineers.

# 8 Terminology

| | |
|---|---|
| FPS | First Person Shooter |
| SDK | Software Development Kit |
| API | Application Programming Interface |
| DIS | Distributed Interactive Simulation |
| HLA | High-level Architecture |
| STS | Saab Training Systems |
| HL2 | Half-Life 2 |
| HL2 DM | Half-Life 2 Deathmatch |
| CoD4 | Call of Duty 4 |
| VBS2 | Virtual Battle Space 2 |
| VTK | Virtual Toolkit |
| C-BML | Coalition-Battle |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| GPI | Game Plug-in |
| UGD | Unified Game Driver |
| UGIM | Unified Game Information Model, the information model created as a part of this thesis. |
| WISE | Widely Integrated System Environment |

# 9 Appendix

## 9.1 Comparison of game SDKs

During development and writing this thesis there where some delays. Internet access was somewhat restricted. Some ports needed to use Steam were blocked by STS firewall. Steam is required to run Half-Life 2 in multiplayer. While waiting for a 3G modem that was going to provide less restricted access to the Internet, STS requested a comparison of SDKs for Half-Life 2, VBS2 and what third party tools are available. This chapter is a comparison between the Valve Source SDK and VBS2 Virtual Toolkit.

### 9.1.1 VBS2 Virtual Toolkit

VBS2 VTK consists of the following tools (Anon, 2008, Virtual Battle Space):

- **Simulation centre** – Modify content (weapon, unit etc.), terrain development (deformable terrain) and scenarios
- **Oxygen 2** – Model configurations and creation. Can import 3DS and FBX support is currently being developed
- **Visitor 4 (includes LandBuilder) –** Lightweight GIS application. Import and modify terrain data. (needs Global Mapper 9 to process raw terrain data)

There are also some significant editor improvements and additions:

- Configuration editor
- Content management
- Team editor
- Weapon editor
- AI force editor
- VBS2 API
- Import 2D topography map to overlay VBS2 map

Some tools needed for VBS2 development are made by third parties:

- **FileBank/PBOBuilder** – Pack VBS2 content into map-files
- **Global Mapper 9** (Anon, 2008, GLOBAL MAPPER) **–** Process raw terrain (converts DTED/VMAP data to XYZ ASCII Grid format. (cost $229-$299 USD)

The Tools have been modified to make generating terrain easier and this is with the MapConfigEditor tool, which helps compile the terrain and sort out shape data. The tools haven't changed much in their application but have been fine tuned to make the process easier.

Virtual Tool Kit is an upgrade to VBS2 in terms of functionality and editor improvements. The updates to the terrain generation are through the VBS2 Development Suite. Visitor 4 allows an easier development path for terrain into VBS2. Oxygen 2 will still be used for creating models; in addition the AI Editor (FSM Editor) will allow AI behavior to be created.

Currently, OpenFlight databases are not natively supported by the VBS2 Virtual Toolkit. Only terrain development from base data (VMAP, satellite imagery, DTED) is currently supported but not natively. Global Mapper 9 is needed to process the raw terrain data. Bohemia is working with Terrex on a VBS2 TerraVista plugin that will support OpenFlight. OpenFlight support for Visitor4 is in development but will not be finished until at least Q2 08. It is possible to import individual OpenFlight models through 3D Studio Max.

The VBS2 tool suite includes the proprietary modeling tool Oxygen 2. It can import 3DS and Maya models (as long as they are optimized for the VBS2 engine).

The largest terrain area so far is a 200km by 200km Afghan AO and is currently limited to approx 2 million objects on the map but Bohemia is working on expanding this limit.

The script engine has been improved and provides a good way to get information in and out of VBS2. The problem with this is that it returns result as strings that need to be parsed. A real API is in development but not available during the writing of this thesis.

**Rapid Mapping in VBS2**

It is difficult and very time consuming to create maps for VBS2. There are courses led by Bohemia Interactive where one can learn to create maps and scenarios, they are several days long and cost thousands of dollars (Lundgren & Perak & Michalski & Ullner, 2007).

### 9.1.2 Source SDK

The Source SDK is free to everyone who owns a copy of the game.
The following tools are provided by Valve (Anon, 2008, SDK Docs – Valve Developer Community):

- Hammer World Editor:
    - Creating level architecture, geometry, texturing, and lighting.
    - Placing models (props) created in 3D modelling packages, such as SoftImage XSI.

    - Placing entities for gameplay.
    - Scripting gameplay entities with Inputs and Outputs.
    - Adding AI navigation nodes for non-player characters.
    - Compiling and running game levels.
- Face Poster - SDK tool used to produce choreographed sequences for the Source engine. It creates and manages facial expressions, lip-sync movements, gestures (body animations), the position of actors in the world, and any map triggers that need to be fired during the scene.
- Model Viewer - program used to preview 3D models created for Source. Can preview and, in some cases, edit many aspects of a 3D model from within Model Viewer.

There are many free third party tools available on the Internet (Anon, 2008, Category: Third Party Tools – Valve Developer Community):

- GCFScape – tool that enables you to browse through the GCF files
- Source Compile Analyzer – a tool which detects errors in compile logs for maps
- The Adobe Photoshop VTF Plug-In –adds the possibility to open and save single-frame/single-face 2D textures directly from Photoshop without the need to convert them to an intermittent format such as a TGA file.
- The 3DSMax VTF Texture plug-in - allows the use VTF format textures in 3DS Max versions 6 to 9 without the need to convert them to another format, such as TGA.
- MapFool – a program that helps with porting HL1 maps to the Source engine
- MDLDecompiler – a tool for decompiling Source models
- StudioCompiler – a tool for compiling models and materials

Hammer World Editor

- Import formats:
    - VMF (Valve Map File),
    - RMF (Worldcraft RMFs)
    - MAP (Worldcraft Map Files).
- Export formats:
    - VFM (Valve Map File)
    - DXF (AutoCAD File)

Creating textures:

- Paint.NET – Freeware, for Windows
- Adobe Photoshop – Professional application for Windows and Mac
- The GIMP – Freeware, for Windows, Mac and Linux

- Pixelmator – Capable Mac application
- any image editor, capable of saving files in the TGA format

**Rapid mapping in Half-Life 2**

It is possible to create models or even entire levels in several 3D modeling software titles and import them to Hammer.

Here are some examples:

CAD:
- Do the work in CAD and save to .DWG
- Import to SketchUp, convert planes into 3D masses and make sure there is no convex geometry. Save to .OBJ (SketchUp Pro)
- See .OBJ to .VMF

Maya (Anon, 2008, Maya to HL2 integration – Facepuch Studios):
- Do your work in Maya and export to .OBJ
- See .OBJ to .VMF

.OBJ to .VMF:
- Import to Blender and export as quake .MAP
- Import to Crafty and export to .VMF
- Open .VMF in Hammer

Valve support model creation with the free modification tool Softimage XSI, it is probably possible to make large parts of the maps in it to.

Softimage XSI
- Softimage XSI can import/export to .VMF
- Open .VMF in Hammer

HL2 to CoD4 (and vise versa) map conversion
- It is possible to convert CoD maps to HL2 maps manually using notepad.

#### *9.1.2.1 Empires mod*

This modification of HL2 is interesting to STS because it combines first person shooter and strategy based game elements. The game is based on two teams that play against each other. Before the game begins the players of each

team vote for one person who will become their commander. The commander has a strategy game type of overview and can create buildings and give orders to the other players in the team. It is the concept of a commander that has a higher overview of the battlefield and gives orders to the soldiers that is the interesting part, because this is the concept of many military trainings systems. Because of this similarity, Empires could be used for certain training.

#### *9.1.2.2 Plug-ins*

HL2 can make use of plug-ins to do certain things. When creating a modification a very large part of the API and classes are available. It is possible to do anything that the Source engine can handle, but a lot must be built from scratch. When creating a plug-in only a small subset of the classes and API is available, but nothing in the game has to be created. There is not very much information which can be extracted from the game using the official API provided by the Source SDK and even less information can be put in than can be taken out.

People in the communities that are using Source based games, have found ways to get access to functions that are not supposed to be accessible from plug-ins. Essentially they are hacks that find offsets to functions in objects that are only accessible when creating a modification. They work by scanning the memory for virtual function signatures in runtime and provide a way to access both members and functions.

**Sourcemm - Source MetaMod**

MetaMod (Anon, 2008, Metamod:Source for Half-Life 2) is a very lightweight plug-in environment written in C++ that sits between the game and Source engine. It intercepts virtual function calls between them and passes C++ pointers to plug-ins [Figure 20 - Overview of HL2 and Source Metamod architectures.]. To use it properly one would have to know what a particular function is used for. There is not very much documentation on the subject, some information can be found asking questions on forums about Source modding. Often things do not work and there is no support from Valve.

**Source Mod**

Source mod (Anon, 2008, SourceMod – Half-Life 2 Scripting) is a scripting plug-in for the Source engine that can be used to write plug-ins without any knowledge of C++. It uses its own scripting language called SourcePawn, although plug-ins can still be written in C++. There is also support for abstracted database access. All plug-ins are loaded into memory and converted into machine code which makes it very fast. Source Mod is open source under the GNU license.

**Mattie's EventScripts**

Mattie's EventScripts (Mattie Casper, 2007, Mattie{info}) are server-side plug-ins that provides an easy to use script interface. There is a list of over a thousand scripts that are officially supported. For more advanced scripting the Python language can be used, and everything that is possible within python can be used. It also supports SQLite database access. There are some problems, most plug-ins seem to be made for Counter-Strike: Source. Half-Life 2 Deathmatch, Empires and Counter-Strike: Source all use different code bases and player classes. Because of this many scripts do not work in Empires and HL2 DM.