Entity Information System

A component of the Sea Traffic Generator

BSc COMPUTER SCIENCE THESIS, IN3405

Version 1.3  June 11, 2012

Authors:

LieYen Cheung  1358219
Nathan Mol  1367757

DELFt UNIVERSITY OF TECHNOLOGY

EEMCS FACULTY

Bachelor Committee:

Mr. ir. Robbert Krijnen (TNO)
Mr. Dr. ir. Gerwin de Haan (DUT)
Mr. Drs. P.R. van Nieuwenhuizen (DUT)
Preface

In order to graduate the bachelor exam at Delft University of Technology, students are required to complete a project that suffices the requirements of the Bachelors of Science project. The project needs to be completed with a group of two to four students and grants the students 15 ECTS, which is about 420 hours. Students can choose from several assignments provided by the university, but students are also allowed to arrange an internship at a Dutch company or even abroad. The purpose of the BSc project is to let students to experience how to independently execute a project and going through the entire software development process.

This thesis is a report of the Bachelor of Science project of Lie Yen Cheung and Nathan Mol, concluding the development of the Entity Information System for TNO Defence, Security and Safety, located in The Hague, The Netherlands. The project spanned 23 weeks starting January 2, 2012 and ending June 13, 2012. The first three orientation weeks are carried out at Delft University of Technology, after which project execution moved to TNO The Hague. This report gives insight in how the project’s development track is carried out, during our internship at an external company.

We would especially like to thank the following peoples for their contribution and support throughout the project:

- Robbert Krijnen (TNO), for being our direct supervisor during the development of the Entity Information System, and giving us advices about the development of the system.
- Roger Jansen (TNO), for helping us with the HLA implementation.
- Eric den Breejen (TNO), for providing the requirements of the system, and feedback on our Requirement Analysis Document.
- Gerwin de Haan(DUT), for being our guiding mentor through the project, and giving us advices to approach the project.
- Peter van Nieuwenhuizen (DUT), for coordinating the bachelor project, and helping us to get everything set up on the DUT side of the project.
- Eric Bouwers, for providing useful feedback on the maintainability of our code.

Furthermore, we also like to thank Ruben Smelik and Phillip Kerbusch for helping us to develop the Sea Traffic Generator during our internship at TNO.

Lie Yen Cheung
Nathan Mol

The Hague, June 11, 2012
Summary

Nowadays simulation is used for many contexts, such as simulation of technology for performance optimization, safety engineering, testing, training, education and video games. Simulation can also be used with scientific modelling of entity systems to gain insight into their behavior and functioning. To develop a realistic simulator, it is useful to work with a virtual world in which a scenario can be build close to reality. But it is very labor-intensive to build a large simulation containing a realistic environment. Even using a scenario generator, which automatically generates a scenario. It can still occur that in a scenario entities are missing or entities are missing some identification information. These missing entities or data may be able to play an important role in the real world.

The Sea Traffic Generator (STG) is a project set up by TNO. The main purpose for setting up this project is because TNO would like to have a virtual world of the sea traffic generated automatically, which is as close to the real world as possible. STG generates the ground truth so behaviors analysis modules (developed by another department) can be tested in a controlled environment before being deployed at the end user. In addition, creating a sketch based environment close to the real world (by hand) requires a lot of effort and time. This condition leads to the need of developing an Entity Information System. When we first have started in January, the assignment we would be working on during our internship was still rather vague. But after several meetings we have finally understood the problem and the assignment that has been waiting for us.

The goal of the project is to develop an Entity Information System, which is able to automatically generate entity information, which can be used for analyze purposes like vessels identification. Our job is to make this Entity Information System, focusing on sea traffic, which generates specific type of identification messages and information about the features of vessels.

Because TNO is an innovation organization, it is expected that the requirements will change over time. We decide to apply the Feature Driven Development (FDD) methodology to our project, having a preference for being able to make changes after the design phase of the program as well, by means of a new iteration. In the FDD development track, the features are split up and grouped in milestones. Every milestone can be changed to support the needs of Entity Information System.

Before working on the first milestone, we have consulted with our supervisor, who has given us a block diagram, containing the high-level design of the entire system. Based on this diagram we have started with domain analysis, by doing literature study and interviewing domain experts. This has allowed us to gain a better grasp of knowledge on the sea traffic domain.

After the domain analysis we have started working of the design of the program. In this phase we have set up the following documents: Requirement Analysis Document, MoSCoW Document and the Architectural Design Document. These documents contain information about the design and requirements which the system should meet.

We have started with the implementation after designing the architecture of the system. During the implementation we have stumbled upon several difficulties. But at the end the difficulties have been solved.
The finished prototype of the Entity Information System is able to handle several inputs based on callbacks and performs actions according to the callback. The system is able to generate live Automatic Identification System (AIS) messages. Additionally, storing data in the database and getting data from the database is also possible. Moreover, the system is also able to repeat certain task based on task and time management. Finally, we have also added a feature generator, which is able to generate a feature set for a vessel, based on the cargo type. The Entity Information System has been designed in such a way that other generators can be easily added in the future without modifying the architecture too much.
# Table of Contents

Preface .............................................................................................................................................. 2  
Summary ........................................................................................................................................... 3  
1. Introduction .................................................................................................................................... 7  
2. Problem and analysis ...................................................................................................................... 9  
   2.1 Description of the problem ........................................................................................................ 9  
   2.2 Tools ........................................................................................................................................ 11  
   2.3 The approach .............................................................................................................................. 12  
      2.3.1 Development methodology ................................................................................................. 12  
   2.4 Deliverables ............................................................................................................................... 13  
      2.4.1 Orientation Report .............................................................................................................. 13  
      2.4.2 Plan of Approach .............................................................................................................. 14  
      2.4.3 Requirement Analysis Document ...................................................................................... 14  
      2.4.4 MoSCoW Document ........................................................................................................ 14  
      2.4.5 Architectural Design Document ....................................................................................... 14  
      2.4.6 Implementation plan .......................................................................................................... 14  
      2.4.7 Testing ................................................................................................................................ 15  
   2.5 Schedule ................................................................................................................................... 15  
3. Designing the system ....................................................................................................................... 23  
   3.1 Global design ............................................................................................................................. 23  
      3.1.1 Global model of HLA ......................................................................................................... 23  
      3.1.2 Global model of the system ............................................................................................... 26  
      3.1.3 Design decisions .............................................................................................................. 28  
   3.2 Milestones ................................................................................................................................ 28
4. Implementation........................................................................................................................................ 42
   4.1 Global system overview .................................................................................................................. 42
       4.1.1 Hla Callbacks ............................................................................................................................ 42
       4.1.2 Distinguish Tasks ...................................................................................................................... 42
       4.1.3 Task handling ............................................................................................................................ 43
   4.2 Classes ............................................................................................................................................... 44
   4.3 Problems, changes and trade-offs .................................................................................................... 46
   4.4 Software Improvement Group feedback .......................................................................................... 49

5. Project evaluation .................................................................................................................................... 50
   5.1 Results ............................................................................................................................................. 50
   5.2 Future work, improvements and recommendations ........................................................................ 57
   5.3 Reflection and conclusion ............................................................................................................... 58

References .................................................................................................................................................. 61

Glossary .................................................................................................................................................... 62

Appendices .................................................................................................................................................. 63
   Appendix A: Sea Traffic generator description (in Dutch Only) ............................................................. 64
   Appendix B: Orientation Report ............................................................................................................ 65
   Appendix C: Plan of Approach ............................................................................................................. 93
   Appendix D: Requirement Analysis Document .................................................................................... 105
   Appendix E: Architectural Design Document ....................................................................................... 144
   Appendix F: MoSCoW Model ................................................................................................................. 173
   Appendix G: AIS messages information .............................................................................................. 178
   Appendix H: Sample of code ................................................................................................................ 183
   Appendix I: MSA Block Diagram .......................................................................................................... 191
   Appendix K: SIG Evaluation .................................................................................................................. 192
   Appendix L: SQL versus NOSQL ........................................................................................................... 194
1. Introduction

As the population grows, and the technology is more complicated, the effort in creating a simulated environment to support military experiments and exercises is going up. Simulation techniques model environments to enable military and analyzers to experiment and exercise the most efficient and effective way of using several (observation) techniques. The scenario generator is a tool that has been used often to fill the simulated environment with objects, with which an experiment is carried out. Unfortunately the simulated environment is not always filled with objects as it should look like in the real world. However, these missing objects can play an important role during the execution of a real operation. At the moment these objects are placed in the scenario manually, which requires a lot of time and moreover, it is a very labor intensive process. This condition leads to the need of developing an object/entity information system, which is able to automatically generate specific entity information.

TNO (Netherlands Organization for Applied Scientific Research) is an independent research organization whose expertise and research make an important contribution to the competitiveness of companies and organizations, to the economy and to the quality of society as a whole. TNO’s unique position is attributable to its versatility and capacity to integrate this knowledge. The daily work of approximately 5400 employees is to develop and apply knowledge. The main office of TNO is located in Delft, but the internship has been done at TNO The Hague. The Sea Traffic Generator project is set up by the Modeling, Simulation and Gaming (MSG) department of TNO. The mission of this department is doing fundamental research in the field of Modelling & Simulation. This means investing and developing new capabilities, improve the effectiveness of existing capabilities. This mission is fulfilled by three closely related key areas of technology research and development services. MSG is characterized by three distinguishing areas of expertise: Simulation Systems Engineering (SSE), Synthetic Environment Modelling (SEM), System & Behavior Modelling (SBM)[1].

Because TNO is an innovation organization, it is expected that the requirements will change over time. Therefore is the Feature Driven Development (FDD) is the most suitable method for the task at hand. Following the FDD method, the required system is split up into various different features, like handling callbacks form several inputs, generating (live or history) AIS data and generating a feature set. This methodology makes it possible to switch feature priority around later on in the development track, as the features necessity changes. FDD is an agile form of developing [2][4]. In the FDD development track, the features are split up and grouped in milestones. Every milestone can be changed to support the needs of the Entity Information System.

The goal of the project is to develop a system, which is able to automatically generate entity information which can be used for analyze purposes. Our job was to make this Entity Information System, focusing on sea traffic, which generates specific type of identification messages and information about the features of vessels.

This report focusses on the development of the Automatic Identification System(AIS) messages generation, and is split up in five Chapters. Chapter two discusses the problem the company presented us with, as well as our analysis. It contains a more detailed description of the assignment and a summary of tools we have used in completing our task, including an argumentation, if applicable. Furthermore, we explain our approach of the assignment and discuss the software
development methodology. We also describe the deliverables and documents regarding to the analyze phase, design phase and implementation/test phase. Each of the deliverable documents will be discussed concisely. Moreover, we also discuss the project’s plan, starting with our initial schedule and ending with how we finally distributed our time during the project and explain why deviations from our original plan occur.

Chapter three briefly explain the design of the Entity Information System and how we divide the assignment in milestones, and we explain each of these milestones briefly. Finally, we discuss our design decisions. When we explain each milestone, we also mention the changes we have made, without going to deep into the implementation. The problems that leads to the changes in the architecture are briefly decribed in this section too, while technical difficulties and limitations occurring in each milestone are described in Section 4.3.

Chapter four discusses the actual implementation of the Entity Information System, starting with a global overview of the system. Then we give a brief explanation of the classes of our implementation. Subsequently, we discuss the problems, changes and trade-offs that we encountered during the system implementation. This section includes an explanation of the changes in the implementation when facing with certain trade-offs or technical difficulties. Instead of explaining the implementation changes for each milestone separately, we have decided to cover all changes and problems per subject and describe it in Section 4.3. This gives a better overview of the problems/changes encountered during the implementation and prevents that we repeat discussing the problems for each milestone again. In the last section we discuss the feedback we have received from the Software Improvement Group (SIG), and how we have processed this feedback to improve the system.

The final chapter, Chapter five, draws a conclusion on the result of the project, including a summary of the final delivered product as well as the developing process. We also give recommendations to improve the system in the future. Finally, we end this thesis with a reflection and a conclusion based on the entire development process.
2. Problem and analysis

This chapter describes the assignment and our analysis of the given problem. First we briefly give a description of the problem, followed by the purpose of the project, including an explanation of the goal of the system. Then we discuss the tools used in developing the system. Furthermore, we discuss concisely how we approach the assignment and have created the documents regarding to each phase of the development process. The last section presents the schedule and discusses the necessary changes we have made with respect to the original project plan to come to our final time distribution.

2.1 Description of the problem

Currently the creation of a realistic scenario to achieve effective training and education or analysis in a virtual environment takes too much time. Until now, the focus lies on the particular behavior, while less attention is paid to fill this worldview, with entities having normal behavior. The lack of it leads to the following problem statement: “generating a realistic view of the world takes too much time” 1. A generated environment is often missing some (identification) information of entities, which could make a scenario more realistic.

The general purpose of the STG project is to create a system, which realizes automatic scenario generation based on entity behaviors. But for the assignment of this project we focus on the maritime that is why the project is called the Sea Traffic Generator.

One of the main goals of the Sea Traffic Generator is to provide a realistic appearance of the real world. It simulates a real world, based on realistic vessel behavior. For validation of analysis algorithms, researchers wants to have a comparison between this simulated real world and the perceived world, the world they can actually perceive by use of sensors. But the vessel behaviors are now generated manually. This is a very intensive process and requires a lot of time. This leads to the development of an Entity Information System, which can add some identification information to entities automatically.

In a (simulation) world a distinction can be made between ground truth and perceived truth. Ground truth refers to information that is collected in a location, for example data of real features and materials on the ground. The collection of ground-truth data could also refer to information that of not within the perceived area of a sensor or user, for example a forest that is located behind a huge mountain. The perceived truth refers to data gathered by the perception of a sensor/user. In the previous example the perceived truth only refers to data about the mountain which is present in the observed area of a sensor/user. It will refer to the forest behind the mountain, while the forest does exist in the area.

During the internship we are responsible for developing an Entity Information System, which is a component of the Sea Traffic Generator and is able to generate specific vessel identification messages, also known as AIS messages. The Automatic Identification System, abbreviated as AIS is an automatic tracking system which is used on ships and by vessel traffic services for identifying and

---

1 The original project description can be found in Appendix A
locating vessels by electronically exchanging data with other nearby ships and AIS base stations. AIS vessel data can mainly be divided into dynamic messages and static messages\(^2\). Dynamic AIS messages contain vessel information like dimension, destination and cargo type. While static AIS messages contain information about like speed, position and course. AIS data is the most important component of the Entity Information System. Since AIS data forms the basis of vessel identification and contain static as well as dynamic vessel information.

The following section gives a global overview of the problem. The figures are screenshots from MarineTraffic [11]. We use it to sketch the problem of project assignment.

Assume the Sea Traffic Generator generates the following scenario:

![Figure 1.0: A scenario generated with STG without identification data.](image)

This is the North Sea, where lots of vessels are sailing through. Each triangular object stands for an active vessel. This means a vessel which is sailing and not anchoring. Each square object stands for an anchored ship. The created scenario looks realistic. But when looking at the details none of the vessels have (specific) identification information. With other words none of the present vessels can be identified. This leads to an unrealistic scenario, since there is no information about any of the ships. To manually specify each ship with identification information requires a lot of time and is a labor intensive task. An Entity Information System can make this a bit easier, by automatically generating AIS data for each vessel. AIS data contain dynamic and static information about a vessel, which can be used to identify a vessel.

Generally, AIS is used on ships and by vessel traffic services for identifying and locating vessels by electronically exchanging data with other nearby ships and AIS Base stations. A scenario which uses the Entity Information System leads to the view in Figure 1.1. Each vessel can be identified, because of the identification data. These data can be used for analyze purposes. For example, observing the behavior of a specific ship in a certain area.

\(^2\) More information about AIS message 123 and AIS message5 can be found in Appendix G.
For vessel identification AIS data is needed. AIS data can be divided into different type messages. But for Entity Information System we only focus on two types of messages, AIS message123 and AIS message5. These two AIS messages form the basis for identifying a vessel, because it contains the position reports as well as ship static and voyage related data.

Our task is to design, implement and test the Entity Information System, which is able to (automatically) generate specific AIS messages for vessels, in order to have a (basis) description of vessels behavior and the opportunity to identify each ship.

2.2 Tools

The Entity Information System uses input from existing systems like High Level Architecture (HLA), VR-Forces and SIMOBS. These are all written in C++, so we are also force to use C++ as coding language. The company wants us to use Microsoft Visual Studio 2008, which ensures compatibility with C++ language and other systems running on Windows.

Since we are not always pair programming, we need a good solution for synchronizing the files. The company has forbidden us to use Dropbox. So we have come with the alternative to use the subversion TortoiseSVN[5], which is also a familiar program for us because we both have used it in the past for other projects. Additionally, the company have arranged a shared folder for us while we are running Windows on the Remote Desktop Server (RDS)[6], not only we but also our supervisor can access to this folder directly. The reason why we use subversion besides the shared folder is because Microsoft Visual Studio 2008 is not running on RDS.

We use UMLet 11.5 [7] for creating UML diagrams, like class diagram, state diagram, sequence diagram, activity diagram. We choose this application because it is a free open-source UML tool with a simple interface but has lots of possibilities and it is very easy to use. Moreover, the created diagrams can be easily exported to images and PDFs, which is definitely a pre for use. Furthermore,

3 More information about HLA, VR-Forces and SIMOBS can be found in Chapter 4 of our Orientation Report in Appendix B.
this application does not need to be installed or whatsoever, just unpacking it to a folder and it is ready to use.

---

### 2.3 The approach

The development of the project is divided into three phases. The first phase is the orientation and analysis phase. In this phase we perform a literature study about the subject and analyze the problem. The second phase is designing the system, according to our research, literature study and requirements; we create the architecture for the system. The third phase is the implementation and test phase, where we actually build the system and test it thoroughly. But since we are using the FDD, the design and implementation phase are iterative processes.

After receiving the project assignment we have started with the orientation phase. This phase is the first phase of development process. We use the first 3 weeks to analyze the problem and to do literature study about the sketched problem. First, we analyze the problem content, to sketch a clear problem description. Then we are studying and analyzing several papers to gather information, which can be used for designing the system. Afterwards, we have investigated the content and functionality of each of the components mentioned on the block diagram, which we received at the start of the project.

We have also analyzed similar systems and make trade-offs, based on the advantages and disadvantages of each system. The result of our research and literature study can be found in the Orientation report in Appendix B. Moreover, we also have several interviews with our supervisor and other TNO employees to set up the requirements for the system. According to the obtained requirements we have created the Requirement Analysis Document. This report can be found in Appendix D. After weeks of research and analysis we have gained the required knowledge to continuing with designing the system. After designing the architecture, we start implementing the system. These two steps are repeated continuously during the project. Chapter 3 describes the design phase while Chapter 4 discusses the implementation.

### 2.3.1 Development methodology

During the orientation phase we had in mind to use the waterfall methodology [3], which is a
sequential design process. The process of the waterfall model can be seen as flowing steadily downwards to each phase from analyzing to designing to implementing and testing. But after discussing with our mentor, it seems better to use an agile software development method, based on iterative and incremental development. This ensures that we have a working prototype, even if not all features are implemented.

We have chosen to apply the Feature Driven Development (FDD) to our project. FDD is an agile form of developing. Following the FDD method, leads to splitting up the requirements into various different features. This makes it possible to switch the priority of features around later on the development of the system, as the requirements and features necessity changes. This will ensure a working version each iteration, which is desirable given the fact that the assignment consists of multiple tasks for the generator. At first, the iterations focus on constructing working, but basic prototype versions of the generator, while later on, new features can be built to improve the functionality of the generator. Every iteration, we decide to work through a full development cycle, taking into account the previous iteration.

After all features are implemented the project will be completed. Failure to finish all features, will leave the system incomplete, but still be able to work. This also illustrates the importance for the order in which features are implements, since there is only limited time in this project.

2.4 Deliverables

As we have mentioned earlier in this thesis, the project can be divided into three phases. According to each phase we have created the related document, to document our results and the process of the project. This chapter gives an overview of all the documents, which are related to our project. We describe each of these documents briefly. Moreover, each of these documents can be found in the Appendices of this thesis.

2.4.1 Orientation Report

The purpose of this report is to do some literature study about the subject of the project and gain more information that can be useful for developing the system later on.

This report contains the result of our research before starting the project at TNO. We have analyzed several papers to obtain information which can be useful for developing our system. At the start we have received a block diagram containing the first thoughts about the functionality of the system. According to this block diagram, we have analyzed each component mention in the diagram\(^4\), to understand more about the functionality of these components.

In the Orientation Report we also mention some information about the tools we are using to develop the system. Moreover, we have analyzed several existing systems and consider these systems, by comparing the advantages and disadvantages of each system. The result of this report can be found in Appendix B.

\(^4\) MSA block diagram can be found in Appendix I
2.4.2 Plan of Approach

Due the amount of time it is necessary to create a plan how to approach the assignment. In our case the assignment is to develop the Entity Information System of the Sea Traffic Generator. Basically this report is a kind of contract which states the problem that need to be solved during the internship. It describes our approach of developing the Entity Information System and contains a global weekly planning of the internship at TNO. Moreover, it states the contract between TNO and us. The result of this report can be found in Appendix C.

2.4.3 Requirement Analysis Document

The requirement analysis document describes requirements of the entire system. TNO requires us to specify each requirement for a general case, a Sea Traffic Generator case and for the Maritime Situation Awareness case. This allows TNO to reuse the documentation of these requirements for other projects. Based on these requirements we have described several scenarios that can occur and have created use cases for each scenario. Based on this document we continue developing our system. The result of this report can be found in Appendix D.

2.4.4 MoSCoW Document

According to the requirements we use the MoSCoW model to filter each requirement in one of the following categories: must have, should have, could have, and will not have. The filtering is also based on the mainly functionality of the system. The MoSCoW model document can be found in appendix F.

2.4.5 Architectural Design Document

This document is created based on the Requirement Analysis Document and the MoSCoW document. The Architectural Design Document gives an overview of the design of the system and our design choices. It describes the architecture of the Entity Information System using various Unified Modeling Language (UML) diagrams and models. Moreover, the Architectural Design Document also mentions how we deal with the data, time and task management and explain some boundary conditions of the system. The result of this report can be found in Appendix E.

2.4.6 Implementation plan

Since it is necessary to complete the project within a certain amount of time, it would be desirable to create an implementation plan, which states when a specific functionality according to the requirement needs to be finished and how we divide the implementation for each milestone. Our implementation plan is described in chapter 2.5 of this thesis.
2.4.7 Testing

To make sure that the Entity Information System works well, it is important to test the implementation thoroughly. So a testing strategy is needed, in order to test the entire system. With each milestone as described in the implementation plan we have created a test suite containing test cases to test the implementation of each milestone. The testing results will be used to improve the system.

2.5 Schedule

At the start of the project we made a draft version of the project’s plan. Figure 2.2 contains the described draft version.

<table>
<thead>
<tr>
<th>week</th>
<th>aantal werkdagen</th>
<th>werkdagen</th>
<th>bijzonderheden</th>
<th>aantal uur</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>di,wo,do,vr</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>di,wo,do,vr</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>di,wo,do,vr</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>tentamens</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>tentamens</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>di,wo,do</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>di,wo,do</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>di,wo,do</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>di,wo,do,vr</td>
<td>voorjaarsvakantie</td>
<td>32</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>di,wo,do</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>di,wo,do</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>di,wo,do</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>di,wo,do</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>wo,do</td>
<td>witte week</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td>witte week</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>do,di</td>
<td>tentamens</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>di,wo,do</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>di,wo,do</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>di,wo,do</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>di,wo</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>di,wo,do</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>totaal</td>
<td></td>
<td></td>
<td></td>
<td>424</td>
</tr>
</tbody>
</table>

Figure 2.2: The draft version of the project's plan

This schedule would meet the requirements that at least 15 ECTS time must be spent on the project. Each ECTS is 28 hours of work, which means we need to spend 420 hours on the project. According to our schedule we spend 424 hours each. This was our original schedule, but due to some circumstances we deviate from it, which we explain later on this section.
Based on the draft version we create a more detailed schedule, which is show Figure 2.3.

The red lines in figure 2.3 stand for a breakweek, which are use for preparing and doing exams. In Figure 2.3 it can been seen that we were planning to use the waterfall model for developing our system. But as we have mentioned earlier our mentor have recommended us to use an agile development method. This leads to a change our schedule. The result is show in figure 2.4.

Based on the schedule in figure 2.4 we have created a more detailed schedule in a table form, the result is shown in Table 2.0.
This schedule only gives a global overview of the planning for the project. After the exams we have thought it would be better to have a more detailed schedule, describing a global planning of the internship. To have everything done before the deadline it is also necessary to have a good schedule, which states what has to be done in that week. That is why we have changed this schedule to a more detailed schedule, which is show in table 2.1. This schedule shown in table 2.1 has worked for us till week 9, where suddenly personal circumstances have occurred, leading to a week behind schedule. But this has been solved by doing more work during the next week.

### Table 2.0: Schedule in table form created on January 18

| Week 1 till week 3: | - Literature study  
|                     | - Analyzing papers  
|                     | - Write orientation report (OR)  
|                     | - Write requirement analysis document (RAD) |
| Week 4 and week 5:  | Break, because of exams.  
| Week 6 till week 8: | From this week working 3 days a week at TNO.  
|                     | - Processing feedback on RAD en OR.  
|                     | - Writing documentation and designing system.  
|                     | - Describe use cases and scenarios.  
|                     | - MoSCoW document  
| Week 8 till week 13:| - Architectural Design Document  
|                     | - Technical Design Document  
|                     | - Implementation plan/schedule  
|                     | - Develop system  
| End of Week 13:    | - First milestone: a basic working prototype (Live AIS generation)  
|                     | - Update the related documents according to the milestone.  
| Week 14            | This week working 2 days instead of 3, because of preparation exams.  
|                     | - Update the related documents according to the milestone.  
|                     | - Improve system by adding new functionalities  
| Week 15            | Break, because of school exams.  
| Week 16            | This week working 2 days instead of 3, because of preparation exams.  
|                     | - Update the related documents according to the milestone.  
|                     | - Improve system by adding new functionalities  
| Week 17 till week 21| - Improve system  
|                     | - Modify documentation according to each milestone.  

This schedule only gives a global overview of the planning for the project. After the exams we have thought it would be better to have a more detailed schedule, describing a global planning of the internship. To have everything done before the deadline it is also necessary to have a good schedule, which states what has to be done in that week. That is why we have changed this schedule to a more detailed schedule, which is show in table 2.1. This schedule shown in table 2.1 has worked for us till week 9, where suddenly personal circumstances have occurred, leading to a week behind schedule. But this has been solved by doing more work during the next week.
After designing the architecture for meeting the first milestone, we have split up the implementation of the system. One of us was focusing on the database implementation, while the other one focuses on the HLA part of the implementation. This has not been planned before so the schedule has to be updated, since we both are working apart from each other\footnote{With apart from each other, we mean that one of us is focusing on the HLA-implementation, while the other one focuses on the database implementation. Each morning we divide the tasks that need to be done that day. Then we individually perform the tasks. Before having lunch and the end of the day we discuss the progress of the performed tasks.}. But we both worked on the implementation and documentation. Additionally, we both have had some problems and technical difficulties.

---

Table 2.1: Detailed schedule created on February 6

---

<table>
<thead>
<tr>
<th>Week</th>
<th>Dates</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-7</td>
<td>06/02 – 17/02</td>
<td>- Continue with work of week 1-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Finishing all related documents of first three weeks</td>
</tr>
<tr>
<td>8-10</td>
<td>20/02 – 09/03</td>
<td>- Designing the architecture of the system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Experimenting with basic database class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Determining the milestones and deadline</td>
</tr>
<tr>
<td>11</td>
<td>12/03 – 16/03</td>
<td>- Implement functions for first milestone</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Writing test cases for milestone 1</td>
</tr>
<tr>
<td>12</td>
<td>19/03 – 23/03</td>
<td>- Implement functions for first milestone</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Writing test cases for milestone 1</td>
</tr>
<tr>
<td>13</td>
<td>26/03 – 30/03</td>
<td>- Implement functions for first milestone</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Writing test cases for milestone 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update documents</td>
</tr>
<tr>
<td>14</td>
<td>02/04 – 06/04</td>
<td><strong>06/04 Milestone 1 done!</strong></td>
</tr>
<tr>
<td>15-16</td>
<td>09/04 – 18/04</td>
<td><strong>BREAK FOR EXAMS</strong></td>
</tr>
<tr>
<td>16</td>
<td>19/04 – 20/04</td>
<td><strong>BREAK FOR EXAMS till Thursday 19 April</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update all documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update milestone 1 according to feedback</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Start implementing features for milestone 2</td>
</tr>
<tr>
<td>17</td>
<td>23/04 – 27/04</td>
<td>- Implement features for milestone 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update documents</td>
</tr>
<tr>
<td>18</td>
<td>30/04 – 04/05</td>
<td>- Implement features for milestone 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>04/05 Milestone 2 done!</strong></td>
</tr>
<tr>
<td>19</td>
<td>07/05 – 11/05</td>
<td>- Update documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update milestone 2 according to feedback</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Start implementing features for milestone 3</td>
</tr>
<tr>
<td>20</td>
<td>14/05 – 18/05</td>
<td>- Optimize code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Clean up code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update all documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Testing system</td>
</tr>
<tr>
<td>21</td>
<td>21/05 – 25/05</td>
<td>- Implement features for milestone 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Work on presentation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update thesis</td>
</tr>
<tr>
<td>22</td>
<td>28/05 – 31/05</td>
<td><strong>25/05 Milestone 3 done!</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Work on presentation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update thesis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Finish all documents</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td><strong>Final presentation</strong></td>
</tr>
</tbody>
</table>
difficulties with our own part of the implementation, but the biggest problem was combining the two parts, which causes problems and errors. This leads to reducing the features in the first milestone and postponing the first milestone (containing all features) to after the exams. Moreover, we have forgotten that the code has also be inspected by SIG which also leads to changing the features of milestone2, by adding some features of milestone 3 to milestone 2.

We had a week delay because we have moved the first milestone to after the exams, but we have worked a full week instead of three days to catch up the missing time, for solving the technical problems and continue with developing the system.

Specific details about the implementation can be found in Chapter 4. Table 2.2 shows the changed schedule, which is constructed on March 29.

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>05/03 – 09/03</td>
<td>- Changing documents after feedback from supervisor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Setting up project with C++ connector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create architecture of database</td>
</tr>
<tr>
<td>11</td>
<td>12/03 – 16/03</td>
<td>- Changing the class diagram, sequences diagram and other UML diagram after receiving feedback on it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Start observing HLA code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implementing basic classes</td>
</tr>
<tr>
<td>12</td>
<td>19/03 – 23/03</td>
<td>- Writing test cases</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Changing architecture of database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implementing HLA code to our project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implementing other classes</td>
</tr>
<tr>
<td>13</td>
<td>26/03 – 30/03</td>
<td>- Implementing other classes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Continue implementing functions of dataset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Continue implementing HLA code to our project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Combining both</td>
</tr>
<tr>
<td>14</td>
<td>02/04 – 06/04</td>
<td>- Combining both and testing it</td>
</tr>
<tr>
<td>15</td>
<td>09/04 – 13/04</td>
<td>BREAK FOR EXAMS</td>
</tr>
<tr>
<td>16</td>
<td>16/04 – 18/04</td>
<td>BREAK FOR EXAMS</td>
</tr>
<tr>
<td></td>
<td>19/04 – 20/04</td>
<td>- Fixing problems</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>20/04 Milestone 1 done!</strong></td>
</tr>
<tr>
<td>17</td>
<td>23/04 – 27/04</td>
<td>- Expanding the functionality of milestone 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Adding new features to milestone 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Changing the ADD</td>
</tr>
<tr>
<td>18</td>
<td>30/04 – 04/05</td>
<td>- Continue implementing new features for milestone 2</td>
</tr>
<tr>
<td>19</td>
<td>07/05 – 11/05</td>
<td>- Update related documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Optimize Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Testing system</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>11/05 Milestone 2 done!</strong></td>
</tr>
<tr>
<td>20</td>
<td>14/05 – 18/05</td>
<td>- Optimize code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Clean up code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update all documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Testing system</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Send code to SIG</strong></td>
</tr>
<tr>
<td>21</td>
<td>21/05 – 25/05</td>
<td>- Continue working on thesis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Work on presentation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Code optimizing</td>
</tr>
</tbody>
</table>
In week 18 our supervisor has received a request from another TNO employee to explore the use and capabilities of NOSQL database and compare this with an SQL databases. The result of this research had to be documented. So we have to handle this request above the work we had. The result of this research can be found in Appendix L.

While doing the final testing milestone 2, before sending it to SIG, we have discovered a problem with the task management, which has to be solved before sending this code to the SIG. This has lead to an update in our time distribution, which is displayed in Table 2.3.

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>30/04 – 04/05</td>
<td>Continue implementing new features for milestone 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Research NOSQL vs. SQL</td>
</tr>
<tr>
<td>19</td>
<td>07/05 – 11/05</td>
<td>Update related documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write report SQL vs. NOSQL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optimize Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Testing system</td>
</tr>
<tr>
<td></td>
<td>11/05 Milestone 2 done!</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>14/05 – 18/05</td>
<td>Optimize code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clean up code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Update all documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Testing system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Work on presentation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Work on presentation</td>
</tr>
<tr>
<td></td>
<td>21/05 – 25/05</td>
<td>Continue working on thesis</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Send code to SIG</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Work on presentation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Code optimizing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Testing system</td>
</tr>
<tr>
<td></td>
<td>26/05 hand in Thesis version 1.0</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>28/05 – 31/05</td>
<td>Processing feedback SIG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Processing feedback thesis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Work on presentation</td>
</tr>
<tr>
<td>24</td>
<td>11/06 – 15/06</td>
<td>13/06 Final presentation</td>
</tr>
</tbody>
</table>

After a meeting with our supervisor, he has also asked us to make an installer of the system. So the installer can run on a “clean” computer. Thus, we have to add this to our initial schedule. Additionally, in the last week of our internship, our supervisor has decided that the program should also have a feature generator. The feature generator generates a set of feature for every vessel. Each vessel type has its own set of feature. So we had to implement this framework, with features too.
The feature generator makes it possible to create a perceived world view\(^6\). This allows a radar system to observe the feature coupled to an entity. For example, radar system X in a certain area observes a ship which is colored red and has a width of 4 meter and has a length of 10 meter.

Adding AIS offset to the system has also been requested. This allows vessels to specify another position for AIS messages compared to its GPS position (e.g. Set AIS position to mid ship instead of GPS receiver location) to the real satellite position. These changes has lead to our final time distribution, which is displayed in Table 2.4.

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>02/01 – 0/02</td>
<td>- Literature study about subject</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Get to know the development environment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Writing orientation report</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Writing Plan of approach</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Gaining requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Writing requirements analysis document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Filter requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Writing MoSCoW document</td>
</tr>
<tr>
<td>4-5</td>
<td>23/01 – 03/02</td>
<td><strong>Break for exam</strong></td>
</tr>
<tr>
<td>6-7</td>
<td>06/02 – 17/02</td>
<td>- Continue with work of week 1-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Finishing all related documents of first three weeks</td>
</tr>
<tr>
<td>8-9</td>
<td>20/02 – 03/03</td>
<td>- Designing the architecture of the system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Experimenting with basic database class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Determining the milestones and deadline</td>
</tr>
<tr>
<td>10</td>
<td>05/03 – 09/03</td>
<td>- Changing documents after feedback from supervisor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Setting up project with C++ connector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create architecture of database</td>
</tr>
<tr>
<td>11</td>
<td>12/03 – 16/03</td>
<td>- Changing the class diagram, sequences diagram and other UML diagram after receiving feedback on it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Start observing HLA code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implementing basic classes</td>
</tr>
<tr>
<td>12</td>
<td>19/03 – 23/03</td>
<td>- Writing test cases</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Changing architecture of database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implementing HLA code to our project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implementing other classes</td>
</tr>
<tr>
<td>13</td>
<td>26/03 – 30/03</td>
<td>- Implementing other classes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Continue implementing functions of dataset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Continue implementing HLA code to our project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Combining both</td>
</tr>
<tr>
<td>14</td>
<td>02/04 – 06/04</td>
<td><strong>Combining both and testing it</strong></td>
</tr>
<tr>
<td>15</td>
<td>09/04 – 13/04</td>
<td><strong>BREAK FOR EXAMS</strong></td>
</tr>
<tr>
<td>16</td>
<td>16/04 – 18/04</td>
<td><strong>BREAK FOR EXAMS</strong></td>
</tr>
<tr>
<td></td>
<td>19/04 – 20/04</td>
<td>- Fixing problems</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>20/04 Milestone 1 done!</strong></td>
</tr>
<tr>
<td>17</td>
<td>23/04 – 27/04</td>
<td>- Expanding the functionality of milestone 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Adding new features to milestone 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Changing the ADD</td>
</tr>
<tr>
<td>18</td>
<td>30/04 – 04/05</td>
<td>- Continue implementing new features for milestone 2</td>
</tr>
</tbody>
</table>

---

*Perceived world view is explained in Section 2.1*
While the table shows some deviations from the original schedule, another change was the use of unit tests. We have written test cases for the first milestone and second milestone, but after cleaning up and optimize the code some test cases cannot be executed anymore, because several functions have been rewritten for optimization and some test cases were covered by other test cases. That is why we do not have many extensive test cases left for the final system, other than the basic test cases like performing database actions and handling several requests. However these basic test cases have helped us to improve functionality of each milestone, because the unit tests were ensuring correct behavior for all methods during each milestone.

Unfortunately we have had to reduce a feature of the second milestone, because we have been depending on the “generating missing AIS algorithm” from other TNO employees, which cannot be finished before the end of our internship. Instead of this feature we have worked on the implementation of handling C2 request, and have integrated this feature within the second milestone.

Based on all mentioned scenarios, it can be concluded that we did deviate a lot from the original schedule, and we did manage to deliver a working prototype of the Entity Information System. However, because the algorithm could not be finished before our internship ends, the feature of generating missing AIS data remained incomplete, while other features were very polished and ready to be used. But we did manage to set up a framework where the missing algorithm can be implemented in the future.
3. **Designing the system**

The design phase is the second phase of the development process. In this phase we create a design of the system. This also means we have to think about the architecture and how we plan to build the Entity Information System, in order to meet the requirements specified in the *Requirement Analysis Document*.

First, we have created a class diagram, based on the requirements and given block diagram, which can be found in Appendix I. Then we have determined the relations between the classes. Then we have created a state diagram and several sequence diagrams to determine the flow of the system. After finishing the documents related to the architecture, we have started to implement the classes and functions as we have described in *the Architectural Design Document*. This design process is repeated for every milestone.

In this chapter we give a short summary of the design of the system. We discuss the global design of the HLA and the system and the design by milestone, with explanations of the design choices. This gives insight in how every feature was designed while keeping the bigger picture in mind.

### 3.1 Global design

In this section we describe the global design of our system. First, we summarize the basic functionality of HLA. Second, we give an overview of the global design with a basic model of how the system works. Third, we explain the relations between the components and discuss the design decisions.

#### 3.1.1 Global model of HLA

As we have described in our Orientation Report in Appendix B, HLA stands for High-level-Architecture. It can be seen as a *communication medium* between different components in a network/simulation. Each component coupled to HLA can subscribe to another component on HLA to receive updates/data from a component. In our case, we subscribed to several components on HLA to received different type of callbacks.

HLA consist of three parts:

1. Rules governing certain characteristics of HLA-compliant simulations.
2. An object modeling scheme that describes the information of common interest to a group (called a federation) of cooperating simulations (federates).
3. The Run-Time Infrastructure (RTI) that provides the software environment needed by the federates to exchange information in a coordinated way.

Entities in a simulation are modeled in the HLA by objects. Each object contains an identifier, state and a behavior description that specifies how the object reacts to states changes. The relationship between the object is specified through:

- Attributes that indicate the state variables and parameters of an object that are accessible to other object.
- **Association** between objects, for example one object is part of another object.
- **Interaction** between objects that indicates the influence of one object’s state on the state of another object.

A federation object model (FOM) specifies the common object model used by all federates. There are four types of events in the HLA: creation of a new object, deletion of an object, a state update, or an interaction.

Figure 3.01 shows a simplified example of HLA. Some components, like Simobs and VR-Forces only publish (specific) information on HLA. They are not interested in data published by other components, so they only have a single arrow. Assume that component X is only interested in data published by Simobs and VR-Forces, while component Y is only interested in data published by Simobs. Then Component X and Component Y must both be subscribed to Simobs in order to receive the published data. But component X also need to subscribe to VR-Forces to receive the data published by VR-Forces. The Entity Information System is interested in callbacks received from VR-Forces and C2, so the Entity Information System component must be subscribe to these two components in order to receive the published requests/callbacks. After receiving a callback/request our system performs an action and sends (generated) data back to HLA if requested. All the (generated) data which we send to HLA are published on this medium. Other components coupled to HLA like C2 can subscribe to these published data. Once they subscribed to these data, they can receive these data each time we publish it. Receive acknowledgement

![Figure 3.01: Global model of HLA with several components.](image)

Another feature of HLA which we use is the built-in time management. The high level architecture time management (HLA-TM) structure supports interoperability among the VR-Forces and our Entity Information System utilizing different internal time management mechanisms. If component VR-Forces publishes updates on HLA, it would be desirable that component Entity Information System immediately, or with a small deviation of time, receives the updates and reacts to it. If this is not the case, the component Entity Information System, might receive the event much later than desired, which leads to a delay. With this delay the system could not operate properly.

HLA-TM guarantees time-stamp ordered delivery of object updates and interactions. Time-Stamp-Order (TSO) Messages are assigned a time-stamp by the generating federate. TSO messages are placed in a queue within the RTI but are not eligible for delivery until the RTI can guarantee that:
- No event will be delivered to the federate with a time stamp less than the current logical time of the federate.
- TSO events are delivered to the federate in time-stamp order, irrespective of the order in which the originating events are sent.

If F is at logical time T, any event generated by F has to have a time-stamp >= T+L; this would allow another federate to advance its logical time to T+L. L is referred to as the lookahead for federate F. The lookahead value represents a contract between the federate and the federation. It establishes the earliest possible TSO message the federate can generate relative to the current time. Lower bound time stamp(LBST) specifies the time of the earliest possible time-stamp-ordered event the federate can receive. Figure 3.02 shows an example of HLA time management with several federates.

![Figure 3.02: An example of HLA time management.](image)

Figure 3.02 contains three federates, the green circles stand for the current logical time position of each federate. The grey squares are lookahead time from each federate. In this figure shows that Federate 2 can increment to the next time step since the resulting time will be within its LBST. However, Federate 3 cannot proceed to the next time step, because it moves beyond its respective LBST value. If Federate 2 sends an event, Federate 1 and Federate 3 receive this event on time t_y. But if Federate 1 sends an event, then Federate1 needs to wait till the lookahead of Federate2 and Federate3 reach time t, before Federate2 and Federate3 can receive the event. This also means that Federate 1 can go to the next time step, till it reaches t_y. Generally, the lookahead of each federate much be covered by the lookaheads of other federates

Federate may proceed to the next time step if it receives the grant that:
- RTI can guarantee that all TSO events with the same time stamp that are less than or equal to t has passed to the federate. Using a method called Time Advance Grant to indicate that the federate’s logical time has been advanced to t.
3.1.2 Global model of the system

Our system basically has four main components. Figure 3.1 shows the basic global model of the system that is going to be developed. In this figure we only focus on the Entity Information System.

The first one is the historyController, which can be seen as a recognizer. This component is responsible for application control. With other words the history controller processes and responds to events, which are usually the result of actions of the user. Whenever a callback is detected, an event is fired to the other components to perform a certain action, based on these callbacks the controller created tasks and put them on the task list. Moreover, the controller also initializes the system.

The second one is the database, which stores the received information. The stored information can be retrieved by other classes. For our system we are using a SQL database and the MYSQL C/C++ Connector in Visual Studio.

The third component is the generator, which defines the incoming information, by adding logic and data relationships. After defining the information, it can be saved using a persistent storage medium, like a database. But it is also possible to use the defined information to perform several actions like generating messages.

The fourth component is HLA, which allows other system to publish information that can be used for performing actions. We use HLA to obtain callbacks from various components. Moreover, we also use HLA for time manangement as mentioned in Section 3.1.1.

![Figure 3.1: Global model of the system](image-url)
As stated before the model seems to be relative simple. There are incoming callbacks, and regarding to the callback an event will be fired to perform a certain action, like defining the incoming data, storing the data in the database, or generating specific data. In order to ensure that the Entity Information System can perform as described above. The system also has to deal with time management, task management and data management\(^7\).

In Figure 3.1 it can be noted that our program does not have a GUI. Because our program is a part of a bigger system, where another component uses our system and shows it in a GUI. For more details of the architecture of the system, we refer to the *Architectural Design Document* in Appendix E.

The general flow of the entire Sea Traffic Generator system is shown in Figure 3.2. The flow starts with building a scenario, and then all entities will be initialized. After that the simulation starts, which is our part of the system. During the simulation state the system can receive different kind of callbacks in order to perform an action. The system gets into the idle state, if the simulation is paused or the system has no task to perform. The system ends in the final state if the program is closed.

![State diagram of the system](image-url)

\(^7\) More about the management of the system can be found in Chapter 3 of The Architectural Design Document in Appendix E.
3.1.3 Design decisions

As we have mentioned earlier, the block diagram\(^8\) can be divided into the main components: C2, VR-Forces and Entity Information System. During the Sea Traffic Generator project we are responsible to develop the Entity Information System component; the other components are be developed by other TNO employees.

We have received this basic architecture at start of the project, described within the MSA Block Diagram, which can be found in Appendix I. According to the block diagram we delve into the architecture of our own part. Keeping in mind, that the system should have a flexible architecture. This means that if a requirement changes, the system should be modified as little as possible. This is also the main reason why we use the basic principles of the Model-View-Controller(MVC) pattern for our architecture.

Using the MVC design pattern allows us to separate the responsibilities of each component, which also means that less logic about other components is required to perform a certain action. With other words the controller receives raw data, without having logic of the data, and passes this data to the generator model which defines it specifically for further usage. Because the generator model does not has any knowledge about the operations, which can be performed using the defined data. It makes it possible to easily add new generators to perform a certain task, without modifying the generator model.

According to the MSA block diagram, the communication between the several components takes place by making use of HLA evolved. But C2 component could also use a SOA to communicate directly with the Entity Information System. For this communication we had in mind to use RESTful Web services, which send simple messages over Hypertext Transfer Protocol(HTTP)\(^{[10]}\). But this have not work out well since we have not considered the amount of data, which can be very large. This makes it difficult and heavy to send the data through a web service. So we have decided to make the database directly accessible for C2, whenever the database contains all of the requested data.

Based on the initial requirements MYSQL database suffices. So we have decided to use the MYSQL database. At a later stage we are asked to consider the use a not only SQL(NOSQL) database, but after some doing research we find out that MYSQL database works out better for our system than a NOSQL database. Other reasons for choosing SQL above NOSQL is the amount of time left for us to develop the entity information generator and the support of the database under Windows. For the database we are using the MySQL Connector/C++ plug-in [8][9]. We have chosen this plug-in because MYSQL meets the needs of the initial requirements. Additionally, it is easy to use and compatible with our system. A more detailed report about SQL and NOSQL database can be found in appendix L.

3.2 Milestones

As mentioned earlier we have worked with milestones, which mean that the architecture and design of the system can change after each milestone. In the sections below we describe the content of each milestone and the architectural changes during and after each milestone.

\(^8\) Can be found in Appendix I
Table 3.2 gives an overview of the milestones. It describes the feature of each milestone briefly.

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Description</th>
</tr>
</thead>
</table>
| **First milestone** | This milestone contains the following features:  
- The system should be able to handle the following two callbacks: handleNewEntityCallback and handleDeleteEntityCallback.  
- handleNewEntityCallback means generating live AIS messages for each new entity. Live AIS generation is the functionality to generate two different type of message.  
  - The first one is AIS message123, which contains dynamic AIS vessel information, like speed, course. This message should be created for each entity every x seconds, where x is the number time determined by the user.  
  - The second one is message5, which contains static AIS vessel information, like destination, dimension, and cargo type. This message should be created for each entity every y seconds, where y is the number time determined by the user.  
- handleDeleteEntityCallback means that the system should stop generating AIS messages for the deleted entity.  
- The generated AIS messages must be stored in the database.  
- Implementation of the HLA layer, which is a class that form the bridge between HLA and the Entity Information System.  
- Database implementation; enable Create, Read, Update, Delete operations.  
- Task management; Keep a task list of all tasks that need to be executed.  
  Live AIS generation is only generating messages from the moment a new entity is created. That is why it is called Live AIS generation. |
| **Second milestone** | This milestone contains the following features:  
- Handling C2 requests. Thus generating (history) AIS messages if needed. C2 could ask for the following data:  
  - Already existing data in the database.  
  - Partly existing data, so get the existing data from the database and generate the missing data  
  - Not existing data. All requested data must be generated.  
- Integrate the system into the updated HLA implementation. The updated HLA implementation contains a built-in time management.  
- Optimize the features of the first milestone.  
- Research SQL versus NOSQL. |
| **Final product** | - Features of the first milestone  
- Features of the second milestone  
- Feature Generator  
- AIS offset  
- Recommendations from SIG |

Table 3.2: An overview of the milestones and system features.

---

9 More information about the two types AIS messages can be found in Appendix G.
3.2.1 First milestone

For the first milestone we have focused on Live AIS Message generation. This means that we should create a program which is able to handle inputs/callback related to Live AIS Message generation. The system should be able to manage callbacks when entities are created and deleted. Based on the callbacks the generator should generate an AIS message for each entity and store the generated AIS message in the database. The latter requires communication with the database, thus a connection must be created to communicate with the database using MYSQL Connector. In additional, it should also be possible to request for generated AIS message, which also needs a connection to the database.

First we have created the class diagram in Figure 3.3 which is based on the global model state in Figure 3.1. The only difference is that we have a new input, C2. This input is not part of the first milestone. This is implemented in a later milestone. Another change is the generator, which is not directly connected to the database anymore, because we have thought that on this way the data management would be more efficient.

In the class diagram it can be seen that we only have four classes. One class for handling database actions, one class for generating AIS messages, one class for handling inputs, and one class for creating AIS objects. We have thought that these classes should be sufficient for the basic generation of live AIS messages. But after discussing with our supervisor it seems that this class diagram is in suffices for decoupling. Because the controller will have too much knowledge about the classes below him, and this is unwanted. Moreover, it does not seem to be as easy as described in Figure 3.3, because the implementation of HLA is pretty difficult and complicated.

![Figure 3.3: First class diagram focusing on live AIS generation](image-url)
After discussing the architecture with our supervisor we have decided that the class diagram has to be changed in order to meet the decoupling requirement. The result is shown in Figure 3.4.

We also have taken the reportGenerator into account when designing the first milestone, this was not actually necessary, since we only need to focus on live message generation.

Figure 3.4 shows a more comprehensive class diagram than in Figure 3.3. We have decided to split up the capabilities of the generator class. Because there can be more generators coupled to the system.

Furthermore, we have extended the functions of the HistoryController class and the database class. Moreover, we have divided the classes into type of categories. Each category has a different color, which states near the class diagram.

There are two types of AIS objects. The first one is message123, which contains dynamic AIS information about a vessel. The second one is message 5, which contains static AIS information about a vessel. But both messages have several common attributes. That why we have made message123 and message5 subclasses from AIS. Additionally, each of the messages should have its own generator, because both messages contain different AIS information[^10].

All generators are subclass of the generator class. The generator class receives an event from the controller to perform a task. Then the generator class sends the task to the specific generator, who has to execute the task.

For the communication between the controller and HLA a HLA layer class has been created, which forms the bridge between HLA and the HistoryController. The HLA layer contains the main functionality, whenever a callback is coming in this class will activate the controller to take an action.

Moreover, we added a class for the Roleplayer actions and a package for handling C2 request.

The functions of the historyController class can be divided into four categories:
1. Catch all incoming requests and keep it up in a queue/tasklist.
   - Sort incoming requests/callbacks and call the functions regarding to the callback/request.
2. Time management: generate AIS and execute task on tasklist.
   - Could be combined with task management for the handling the following inputs:
     - HLA: assign task to generators that are able to handle the task.
     - C2: assign task to specific responsible component (generating or storing AIS).
     - RolePlayer: perform a specific task.
3. Return requested data to the component requested for it. These components are:
   - HLA, which receives the generated AIS message with a certain frequency (depending on generator)
   - C2, which received the request data as soon as possible
   - RolePlayer, does not receive anything.
4. Keep a list of available and on HLA existing entities.

[^10]: More information about the two types AIS messages can be found in Appendix G.
The functions of the database class are storing the specific generated AIS messages by the message type. Additionally, the class should also be able to get specific requested AIS messages from the database. The same goes for the INTELL Report and the Alpha report.

Figure 3.4: Class diagram created on March 08
After integrating this class diagram into our project, we have thought that the architecture for the first milestone has been finished. This system is capable to handle HLA callbacks, and generates live AIS messages, which can be stored in the database. It is also possible to request the generated AIS messages from the database. But after a discussion with our supervisor, it seems that the generator class has too much knowledge about the generators. So if they wanted to add a new generator, a lot of changes should be made in the system. This is not what they want. So we have to redesign the architecture on such a way that the generator cannot have too much knowledge about the generators but still be able to perform certain actions. With other words the idea of decoupling should be taken into consideration when redesigning the system.

This leads to changes the architecture of the system, which is shown in Figure 3.5. We have made the generator class abstract and added some classes GeneratorModel, Task, TaskRef. Moreover, we have added a package delta_t which is the core of handling taskmanagement. We have dropped the reportGenerator because it was not necessary for the first milestone. Furthermore, the functions of the each class have extended because of these changes.

The Task class creates task objects for every task that needs to be executed. The class TaskRef provides the link between the Delta_T list and the Task Objects.

The class GeneratorModel is responsible for initializing all generators and providing data to all generators, without having any knowledge about the generator’s responsibilities.

Due the size of the class diagram, the class diagram has been cut into two parts. The first part is shown in Figure 3.5a and the second part is shown in Figure 3.5b. Together they describe the architecture for the first milestone.
Figure 3.5a: part one class diagram created on March 22
Figure 3.5b: part two class diagram created on March 22
3.2.2 Second milestone

Based on the results of the first milestone and lack of implementation experience in C++, we have changed the some parts of the architecture to optimize and expand the functionality of the second milestone.

The second milestone has been extended with capabilities of the first milestone and handling C2 requests, focusing on AIS history message generation. Moreover, the new HLA implementation update containing a built-in timer instead of a sleep function, and time management update should also be included in the second milestone.

We use enumeration to determine the type of several inputs, boundary and messages. The functions in the history controller have been extended to meet the new HLA implementation. The database class has also been extended to meet the features of milestone 2.

C2 could request the following data:

4. already generated data, stored in the database.
5. data which is not completely generated. Thus the missing AIS data should be generated.
6. Not (already) existing data. Thus the whole requested interval of AIS data should be generated.

To optimize the tasks in the system we added four classes, which are subclasses of the class task.
- The first class is appearance task, which is a task to determine the appearance of a vessel.
- The second class is C2Subtask, which creates tasks for handling C2 requests.
- The third class is C2RequestTask, which creates tasks for handling C2 requests.
- The fourth class HLATask, which created task based on HLA input.

Furthermore, we have created a class convert, which contains all the convert methods used in almost every class. We have moved most of the convert functions in each class to a general convert class. This has lead to efficiently increment and giving a better and clearer overview of the code.

The above mentioned changes have lead to the class diagram of the second milestone, which can be found in Figure 3.6a and Figure 3.6b.
Figure 3.6a: part one of the class diagram belonging to second milestone.
Figure 3.6b: part two of the class diagram belonging to second milestone.
3.2.3 Final product

After the second milestone we have optimized the code in order to send it to SIG. We have moved some functions to other classes and have improved the task/time management implementation, after finding some bugs. Additionally, after a discussion with other colleagues the unique user ID of a vessel has changed to a string type instead of an integer. This has also been changed. Furthermore, we have also added a class XML, which is able to read an XML file containing the database configuration. This makes it easier for the user to modify the database configurations. We have also added a class Global, which defines the properties of the database, regarding to the configurations read with the XLM reader. Otherwise, if not defined it uses the default values.

We also have received the request to implement a feature generator, which generates a set of features according to the cargo type of the vessel. For example, when a new vessel is initialized, the feature generator generates feature like color, material and height, offset for this vessel. These features have to be saved in the database. The reason for implementing a feature generator is to create a perceived world view\(^\text{11}\), so a specific radar system could perceive a certain feature. For example a radar system X could perceive that a vessel has a height of 5 meters and the ship is colored red, by receiving feature data of a vessel.

The request has also been to implement an offset for AIS position, which allows vessels to specify another position for AIS messages compared to its GPS position (e.g. Set AIS position to mid ship instead of GPS receiver location) to the real satellite position. We have combined this feature using the feature generator.

These changes have lead to the class diagram in Figure 3.7. This class diagram does not specify the functions of each class, due the amount of space. But it does describe the whole architecture of the system which we have developed. To see the whole class diagram with functions of each class, please go to Appendix E Architectural Design Document section 6.4.

A briefly explanation of the class diagram:
HLA has been used to receive input from VR-Forces/C2; these inputs will be processed by HLA layer (hlafederate), which sends the input/callbacks to the historyController. Based on the input and callbacks the historyController used classes in the taskmanagement to created task that needs to be executed.
Then the historyController can send an event to the generatorModel and C2Model to perform a certain action with a generator based on the tasks in the tasklist. But the historyController can also save data directly in the database. It is also possible to save generated data into the database.
The database uses the globals for configuration of the database properties. The global class uses an XML reader to read the configuration and define them in the globals.
The class convert contain convert functions for all class, so we have not draw any arrows to this class because it does not increase the overview of the class diagram. But keep in mind that this class has been used by almost every other class which needs to convert data. The same goes for the Enumerators package. The package Enumerators has been used to define some enumerators which are used to determine what the type is of specific data, like task type.

---

\(^{11}\) Perceived truth is explained in Section 2.1. Based on the perceived truth a perceived world view could be made.
After receiving feedback from SIG we have needed to update the architecture in order to meet the recommendations of SIG. Moreover, during the final testing we have also discovered another problem of the system, which is the bottleneck of storing data. A storing data operation takes time, and if there are lots of entities (read 2000), than it takes a lot of time before all vessel can be initialized and stored the generated data. So we need to optimize the system. This lead to the class diagram shown in Figure 3.8, which is also the final version of our class diagram. This class diagram

---

12 More details about the SIG feedback can be found in Chapter 4.5.
does not specify the functions of each class, due the amount of space. But it describes the whole architecture of the system which we have developed. To see the whole class diagram with functions of each class, please go to Appendix E Architectural Design Document section 6.4.

We have added a class C2Model, which contains functions that were in the database class and performs functions for executing C2 tasks. C2Model adds logic to C2 tasks. Moreover, we have moved some cut some functions into separate functions to reduce the complexity of functions. We have also improved the code to optimize the store operations. By performing store operations in a batch, which means instead of performing 10 store operations, the system now performs one store operations containing all conditions of the 10 store operations. Moreover, all tasks are now distributed evenly. Finally, we also have cleaned up and rewritten some codes to improve the functionality and maintainability of the system.
4. **Implementation**

The implementation is the third phase of the development process. In this phase we have started building the system, by writing codes based on the created architecture and previous literature study. In this chapter we give a briefly system overview and explain the responsibilities of each class of our implementation. Then we discuss what problems arose during the process and how these problems were solved. It also reflects on changes that were made, considerations and problems that were encountered. Next to that, we also discuss the feedback from the Software Improvement Group (SIG) and the improvements that have been made regarding to the feedback. Note that we do not discuss the changes of each milestone separately. We combine the changes and problems per subject and discuss them in Section 4.3.

4.1 **Global system overview**

In order to explain how the complete system works, we first need to determine the dependencies of the system. Our system depends on the output of VR-Forces, or in other words, dependent of HLA callbacks. To categorize and correctly handling these types of inputs from HLA, we have to introduce tasks, which in turn are correctly handled by the task management. Detailed description about the way we handle task management, can be found in Appendix E, chapter 3.2 and 3.3.

4.1.1 **Hla Callbacks**

Our system is based on various callbacks derived from HLA. The main callbacks are; “new entity x has been initialized on HLA”, “entity x has been deleted from HLA”, “Hi, I am C2 and I have a database request y, is requested interval z complete? If not, generate!”. These callbacks lead to the development of tasks.

4.1.2 **Distinguish Tasks**

As said we have to develop tasks based on their input, but also we have to make sure that the tasks result as we please. An overview of the tasks is presented and briefly explained each task.

4.1.2.1 **HlaTask**

HLATask Handles live AIS message generation and Feature generation for a specific entity. This task is based on the new entity callback and deletes entity callback both from HLA. When a new vessel appears, we need to do **something** with it. Directly when a vessel disappears, we must do **nothing** with it.

4.1.2.2 **AppearanceTask**

AppearanceTask unsers the entity appearance registration in the database and local administration (HistoryGenerator) for a specific entity.
4.1.2.3 C2RequestTask

C2RequestTask handles a C2 Request-callback containing an interval from one or more SQL table names and one or multiple entities. After performing the task, a notice will be send back to HLA.

4.1.2.4 C2RequestSubTask

To handle subtask of a C2 Request to fill a specific gap (see Section 4.1.3.4). A vessel could first occur in an area and later move out of the area and coming back to the same area later on. So it is necessary to determine when a ship occurs and leaves an area. The missing data between the time the ship leaves an area and the ship enters an area, are we calling gaps.

4.1.3 Task handling

After distinguishing between the various possible tasks, we need a proper handling of the tasks. While simulating, we use the time/task management as described in The Architectural Design Document section 3.2 and section 3.3. When a new task needs to be executed, it first needs to determine its type of task (HLA, Appearance, C2Request). After that each task needs to be performed its own way.

4.1.3.1 Handling HlaTask

A HLA task contains a generator which needs to generate data for a given entity. The possible generators are: LiveMes123Generator, LiveMes5Generator, FeatureGenerator. These are respectively responsible for generating AIS message 1, 2, 3, AIS message 5 and random feature generation derived from a data set. The LiveMes123Generator and LiveMes5Generator both get their information (if existing) from HLA and sends it back to HLA after the message creation. All three generators also saves the generated data to a database.

Live AIS message generation is responsible for keeping records of the messages periodical. To realize this, the generator schedules a new task after completing the task with a defined heartbeat (impulse).

4.1.3.2 Handling AppearanceTask

The Appearance task is mostly introduced to keep a record of the entity administration like: “this ship is (not) currently present at HLA”, “what time did this ship (dis) appear”. This task is mainly introduced to have a proper handling of the availability of a specific ship at a certain time. After updating the administration, the system is able to initialize the entity using the GeneratorModel. In order to ensure equal distribution, the start delta time is first determined. The task is also used to keep a record of the appearance intervals.

4.1.3.3 Handling C2RequestTask

When someone wants to obtain information about an entity on a specific interval, this must be handled properly. The C2RequestTask will therefore using some functions to determine gaps in our live generate data, using the appearance intervals (see 4.1.3.2 Handling AppearanceTask). After determining the possible gaps in our generated data, a C2RequestSubTask is added for each gap with calculated start time, number of recursions and the specific entity identification. Appendix H shows some code samples which are use to determine the gaps.
4.1.3.4 Handling C2RequestSubTask

As said above, these subtasks are used to fill a gap. These tasks use GapMes123Generator and GapMes5Generator in order to generate and save these empty intervals. These generators make use of the start time and the number of recursions. The messages are stored after generating them using a backtracking algorithm. This algorithm is currently not finished yet.

After generating data for the given time and recursion, a new task will be created with one reduced recursion, the start time is also decreased with a heartbeat. After completion of all (number of recursions) tasks the gap is filled. Proper subtask administration is needed to determine the completion of a C2 Request.

4.2 Classes

In this section we briefly describe each class of our system to sketch how our system works. In Figure 4.1 you can see the packages of our system, where each package contains one or more classes.

We will describe each of the following packages starting from the (History) Generator package. We will not describe the HLA package, because this package is generated by another employee of TNO.

As can be seen in figure 4.1 the HLA-Layer Package is “half inside” the Entity Information System, because this package forms a kind of bridge between the HLA package and the (History) Generator package.
(History) Generator
This package contains all classes that we have implemented for the system.

- **HLA-Layer**
  - HLAFederate
    This class is a bridge between HLA and (History)Generator. It also contains time management.

- **Controller**
  - HistoryController
    This class is responsible for controlling all classes below it. It also handles callbacks from HLA. It includes task management.

- **Database**
  - DBHandler
    This class is responsible for handling database actions like storing data in the database, performing requests from other components, and cleaning up databases.

- **Others**
  - Convert
    This class contains convert functions that are required for all other classes. Like converting `IntegerToString` method.
  - XML
    This class is able to read an XML file containing configurations for the database.
  - Global
    This class defines the configuration and properties of the database.
  - Enumeration
    This package contains classes to determine the message type, input type, and query type.

- **Model**
  - GeneratorModel
    This class is responsible for providing the right job(s) to one or multiple generators. It is also responsible for initializing all generators based on callbacks.
  - C2Model
    This class is responsible for adding logic to C2 tasks and assigning jobs to one or multiple generators.

- **Generator**
  - LiveMessage123Generator
    This class is responsible for generating AIS messages 1,2, and 3.
  - LiveMessage5Generator
    This class is responsible for generating AIS messages 1,2, and 3.
  - GapMessage123Generator
    This class is responsible for generating gaps from an interval for messages 1,2, and 3.
  - GapMessage5Generator
    This class is responsible for generating gaps from an interval for messages 1,2, and 3.
  - FeatureGenerator
    This class is responsible for generating features for a vessel.

- **Objects**
  - AIS
    This class is responsible for creating an AIS Message.
This class is responsible for creating an AIS Message 1, 2 and 3.

- Message 5
  This class is responsible for creating an AIS Message 5.
- Feature
  This class is responsible for creating a feature.

- Task Manager
  - Delta_t
    This class forms a framework containing task management structure of the system.
  - TaskRef
    This class forms a reference between task and delta_t.

- Task
  These classes have been described in Section 4.1

### 4.3 Problems, changes and trade-offs

In this section we discuss several problems that we encountered during the project and implementation of the milestones. We briefly explain the problem and how we have solved it. Additionally, we explain why we have made several changes and trade-offs. Instead of explaining the implementation changes for each milestone, we have decided to cover all changes and problems per subject and describe it in this section. This gives a better overview of the problems/changes for each subject encountered during the implementation.

**Enumeration**

We use enumeration to categorize the type of objects. This was necessary, for example, to determine what the type the callback is. In order to stimulate the generators to perform a task if they can handle such input. The same applies to the different type of messages and boundaries of an interval. It is possible to reach the same effect by manually implementing string and comparing these values. But using enumerations makes it possible to realize decoupling, because one can change the name of an enumeration without affecting the rest of the code.

**Database**

- Connection:
  First idea was: the system only creates a connection with the database if the database needs to perform an action. And after the action the connection with the database would be closed. Because we have thought that this way would be the most efficient. Unfortunately this has not worked out perfectly. After several test cases it appears that the connection can be lost and the performed action failed. Due the amount of continuously opening and closing the connection. So we have decided to connect to the database when the program starts. This has not the perfect choice, because a database action can be executed before the connection has established. So afterwards we have decided that the system should connect with the database while the system is initializing. After lots of attempts the connections performs correctly.
- Storing data:
  First we have used a list containing string objects of an AIS message to save data to the database, because we have thought it is easier to excess and update data since it is not necessary to convert any of the data. Later, we have decided to change this to a map containing pairs, but this was not efficient, since data can be lost after casting/converting. Afterwards we have decided to send AIS objects to the database, which reads out all the information of this object and save it into the database. This is more efficient and faster than the previous ways. But in the final version we have decided to store AIS by conditions, which mean the system builds conditions set and save this conditions set directly to the database. This definitely increases the performance of the system, since executing a SQL query causes a delay.

Requesting:
We have decided to return a list containing all requested information with string objects, because we save the data as string objects in the database. So getting the data from the database should be a piece of cake. But in this case we have not though about the amount of data. This can be a problem if we send the requested amount of data through the network. Afterwards we have decided to return a ResultSet, containing the requesting information. Since the size of such a set is much smaller en easier to handle for the requested person.

In the final two weeks we have decided to rewrite the database class completely, because is cleaner to have one general method for storing data. So we have changed the store methods to a general method storeDataByConditions, with stores the data based on the given condition set. On this way the database would not have too much knowledge about the data that needs to be stored. The same goes for the function getting and deleting data from the database. Moreover, we have moved the exception messages retrieved from errors to a separate function, because this gives the code a clearer overview.

Soap REST
To enable C2 doing a query request, we have in mind to use a server-client approach called REST SOAP. This can simplify sending the data request and easily returning the data in an XML format. Theoretically this should work, but after installing and implementing a variant called gSOAP it seemed less efficient than originally thought. By a number of reasons: a database request can return a large set of data, is the system able to handle this? During the implementation it has turned out that the link between our program and the server has been more difficult to achieve than we have expected, and this can bring some problems with prospect to the deadline. An alternative solution is found: centralize the database, so the database is (directly) accessible to C2, after sending a request to our system using a HLA callback. Once the request data is ready we send a message with a request (SQL query) and an acknowledgement.

C/C++ connector
To continually store a large amount of data it is convenient to use a database. Our choice directly fell to MYSQL, which is suitably qualified for storing large amount of data and deal with continuous stream of read and write operations. Furthermore, our background knowledge have also played a significant role. We need to find a connector which is able to perform Create, Read, Update, Delete (CRUD) operations. After a bit of research we found the C/C++ connector provided by MySQL itself.
After experimenting with the basic functionality of the C/C++ Connector, we have come to the conclusion that this is an appropriate way to perform database operations. In addition, we have had problems between HLA and the library of the C/C++ Connector, caused by overlapping function calls. But this problem can fortunately fairly easy be solved by deprive the namespaces of all classes.

Later, we have been confronted with strange problems such as loss of the database connection and calling an empty connection object while the object was previously created. After extensive testing we have come up with the problem and its solution, outlined in the section Task management.

Task management
The importance of using a task manager for all possible tasks became clear to us. During the first milestone of the program we do not have much different tasks to be fulfilled. The only task we had was generating and saving an AIS message. As the development makes our program crashes regular, we have been confused about why this could happen. After testing the program thoroughly, we have found out that we have given every new callback directly permission to access the database class, in order to execute create/read/update/delete operations. But the database class cannot handle multiply callbacks at the same time, since we only use one database handler object. We have found that the problem was occurring because one or more callbacks tried to reach the database handler while the database handler was still performing an action. The solution for this problem was rather simple: add every possible action to the task management, so that each task is neatly and sequentially executed.

Time management
Since we are dealing with multiple federates in a network, it is necessary to handle time management. It enables that each federate works relative synchronizes to each other. If this is not the case, then anything that probably can go wrong might go wrong. Therefore, we use HLA with a built-in time management, which ensures that our Federate only allowed doing something for the next time step as the slowest Federate the network is ready for the current time step.

Memory leaks
Another problem that occurs together with the previously described problem is the problem caused by memory leaks. We have received weird notifications, from the database handler, like it was not possible to connect to the database, since some parts of the connection have been damaged. After investigation we have found that newly created objects have not been properly removed, we have removed the objects but we did not set the pointer value to zero, which stands for empty. We have solved this problem by using a SAFE_DELETE method, which deletes the object and sets the pointer of that object to zero.

HLA
After an update of the generated code of HLA, it seems that our program is containing a bug. But after testing this with several test cases, it became clear that the bug was not occurring in our program. The problem was that a specific callback from HLA has not defined correctly. Based on this problem we have learned that it is always necessary to test the coupling with other program thoroughly, before concluding something.
Type definitions
We have used type definitions to define several complicated data-structures. This gives faster insight into what happens in the code, it also ensures that less code is needed and has to be read.

Role player
Based on our initial observation, it seems practical to make use of a role player, who can manually manipulate the data and simulation speed. But after all it have seemed that the role player is superfluous, because it should be possible to manipulate all data with VR-Forces. But till then HLA is responsible for modifying the simulation speed.

4.4 Software Improvement Group feedback
In this section we discuss the feedback provided by the Software Improvement Group (SIG). The code has been evaluated and rated by them. The received feedback can be found in appendix K. After the first feedback we have send the code again with improvements, based on the feedback.

First feedback
According to SIG, the rating of our code was more than 3 out of 5 stars on their maintainability model. Which means that the average code maintainable. The score has been brought down by the Unit Size, the duplication and lack of components.

Based on the feedback we have improved the Entity Information System so it meets the recommendations of SIG. We split up some methods into separate functions to reduce the unit size of functions. Moreover, we have tried to reduce the amount of duplications of functions, but due the handling of different message we cannot reduce all duplicated codes. Some of the methods for handling the different type of message might be almost the same, but each of the classes should be seen as an individual object. The system should still be able to perform actions if one of the messages is decoupled from the system. In our project we have divided the classes into components, but for uploading to SIG we just created a folder named “src”, and copied our source code into that folder. That is why the uploaded file-system did not have a component structure. For the next submission we will changed the folder based on our visual studio project.

Second feedback
After improving the code based on the feedback, we have uploaded it to SIG again for another review. From the received feedback is appears that the code have scored 4 out of 5 stars. This increase is attributable to a slight increase for Unit Size and Complexity. According to the evaluation of SIG, we have processed most of the recommendations to improve the maintainability of our system. There might still be some duplication in the code, but we have not made it to remove all of them, due the dependent relationships between the components and new added codes. Moreover, while we expanded the system, the amounts of test cases have also been increased. SIG also mentions this as a plus point.

13 Also see requirement Analysis Document requirements REQ_GRP_001 - REQ_GRP_004
5. **Project evaluation**

In this chapter we revisit the assignment defined in the introduction and chapter 2, by reviewing the schedule of requirements that we have composed for every milestone. It uses the previous chapters to draw a conclusion on how we finished the project, with regards to the problem description. After that, it will offer some ideas for future work to improve the final delivered product, as well as discuss some recommendations for new generators. Finally, there will be a reflection, describing what we have done differently if we start all over again. Moreover, we also describe a short reflection on how we experienced the project in general.

5.1 **Results**

The STG project is about the problem statement: “generating a realistic view of the world takes too much time”, focusing on sea traffic. A generated environment was often missing some (identification) information of entities, which can make a scenario more realistic. This sub problem has partly been solved by creating an Entity Information System, which is able to automatically generate AIS message for vessels. Moreover, the system is also able to generates a features set for each vessel. The solution meets the following requirements:

- **REQ_GEN_009**: The system must store entity information in a data store.
- **REQ_SHI_001**: The system should be able to store information of the ships.
- **REQ_HMV_004**: The system must be able to store position(s)

The prototype of the system is able to store AIS data, appearance data and features of a ship in a database. The positions of a vessel are stored within the AIS data. The stored data can be requested using database requests.

- **REQ_GEN_010**: The user must be able to request information from a data store using queries.
- **REQ_SHI_002**: The user must be able to request data from an information-system using queries.
- **REQ_AIS_001**: The user must be able to request AIS information for one or multiple entities.
- **REQ_HMV_003**: The user must be able to request stored position(s) of the ships

Data from one or multiple vessels can be requested by performing request to the database. If information about the vessel is missing, the generator would generate the missing data.

- **REQ_GEN_011**: The system must be able to initialize a scenario based on a random start time

It is possible to run the simulation with a random initial time. The starttime can be modified by VR-Forces or manually in the program.
• **REQ_GEN_015**: The system must be able to accelerate the simulation speed.

• **REQ_GRP_004**: The user must be able to manage the simulation speed.

The system enables the user to change the simulation speed, to avoid long waiting before an event occurs. The system is able to accelerate the simulation speed, if the slowest task can handle it too.

• **REQ_GNT_001**: The system must have an Entity generator.

This requirement is partly covered, since the system only has an AIS generator and an GAP generator and a feature generator. But more generators can be added to the system. The entities, in our case the vessel, are created by VR-Forces.

• **REQ_GNT_002**: The entity generator must have a varying output.

The system stores data to the database, depending on the input of HLA. If the input from HLA differs the output in the database will differ, but if the input from HLA remain the same than the output in the database remain the same. Based on the created entities the Entity Information System generated specific information, like AIS, appearances and features.

• **REQ_GRP_001**: The user should be able to add entities manually.

In this case a vessel can be added manually by accessing the database manually and perform add “new entity action”. Only the assigning DMP to the entity does not work in our system, which led to partly covering of this requirement.

• **REQ_SHI_003**: The system should be able to generate history, based on DMPs.

• **REQ_AIS_002**: The system should be able to generate specific AIS information about an entity.

• **REQ_HMV_005**: The system must be able to generate past position(s)

This requirement is partly covered. The developed system is able to (automatically) generate (history) AIS messages, but with the default values. Unfortunately, the generating history algorithm cannot be finished before our deadline. But the system is able to generate live AIS messages for each entity, static AIS information as well as dynamic AIS information,

• The system must be suitable for the decoupling idea and should be developed to increase reusability.

The generator system is designed in such a way that new generators can be added easily without modifying too much in the code. This has been shown by implementing the feature generator in the last two weeks of the project.

We have implemented most of the MUST requirements which are related to our part of the entire system. Furthermore, we have also implemented some “should” requirements, which are related to the (history) generator and stated in the MoSCoW Model.
All together, this project has brought a basic functional prototype of the Entity Information System. This comes with elaborate documentation of the system, including design document and UML documents like class diagram, sequence diagram, flow diagram. The features that have not made it to this project can still be implemented in the future, if wanted, which will be discussed in the next section.

The extended CPPUnit test classes that have been written after each milestone have been helped us to find bugs and problems during the development of new milestones. It adds up to the maintainability and expandability of the system as a whole.

The following figures show some results of our Entity Information System.

<table>
<thead>
<tr>
<th>FEATURE_HISTORY_ID</th>
<th>ENTITY_ID</th>
<th>TIME_IN_SEC</th>
<th>CARGO_TYPE</th>
<th>FEATURE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>151</td>
<td>M203</td>
<td>1337759502</td>
<td>1</td>
<td>COLOR</td>
<td>PINK</td>
</tr>
<tr>
<td>152</td>
<td>M203</td>
<td>1337759502</td>
<td>1</td>
<td>MATERIAL</td>
<td>FOIL</td>
</tr>
<tr>
<td>153</td>
<td>M203</td>
<td>1337759502</td>
<td>1</td>
<td>HEIGHT</td>
<td>982</td>
</tr>
<tr>
<td>154</td>
<td>M284</td>
<td>1337759502</td>
<td>1</td>
<td>COLOR</td>
<td>RED</td>
</tr>
<tr>
<td>155</td>
<td>M284</td>
<td>1337759502</td>
<td>1</td>
<td>MATERIAL</td>
<td>ALUMINUM</td>
</tr>
<tr>
<td>156</td>
<td>M284</td>
<td>1337759502</td>
<td>1</td>
<td>HEIGHT</td>
<td>619</td>
</tr>
<tr>
<td>157</td>
<td>M656</td>
<td>1337759502</td>
<td>1</td>
<td>COLOR</td>
<td>GREY</td>
</tr>
<tr>
<td>158</td>
<td>M656</td>
<td>1337759502</td>
<td>1</td>
<td>MATERIAL</td>
<td>Steel</td>
</tr>
<tr>
<td>159</td>
<td>M656</td>
<td>1337759502</td>
<td>1</td>
<td>HEIGHT</td>
<td>110</td>
</tr>
<tr>
<td>160</td>
<td>M608</td>
<td>1337759502</td>
<td>1</td>
<td>COLOR</td>
<td>BROWN</td>
</tr>
<tr>
<td>161</td>
<td>M608</td>
<td>1337759502</td>
<td>1</td>
<td>MATERIAL</td>
<td>WOOD</td>
</tr>
<tr>
<td>162</td>
<td>M608</td>
<td>1337759502</td>
<td>1</td>
<td>HEIGHT</td>
<td>362</td>
</tr>
<tr>
<td>163</td>
<td>M426</td>
<td>1337759502</td>
<td>1</td>
<td>COLOR</td>
<td>RED</td>
</tr>
<tr>
<td>164</td>
<td>M426</td>
<td>1337759502</td>
<td>1</td>
<td>MATERIAL</td>
<td>FOIL</td>
</tr>
<tr>
<td>165</td>
<td>M426</td>
<td>1337759502</td>
<td>1</td>
<td>HEIGHT</td>
<td>814</td>
</tr>
<tr>
<td>166</td>
<td>M386</td>
<td>1337759503</td>
<td>1</td>
<td>COLOR</td>
<td>GREY</td>
</tr>
<tr>
<td>167</td>
<td>M386</td>
<td>1337759503</td>
<td>1</td>
<td>MATERIAL</td>
<td>TRIPLEX</td>
</tr>
<tr>
<td>168</td>
<td>M386</td>
<td>1337759503</td>
<td>1</td>
<td>HEIGHT</td>
<td>522</td>
</tr>
<tr>
<td>169</td>
<td>M152</td>
<td>1337759503</td>
<td>1</td>
<td>COLOR</td>
<td>RED</td>
</tr>
<tr>
<td>170</td>
<td>M152</td>
<td>1337759503</td>
<td>1</td>
<td>MATERIAL</td>
<td>WOOD</td>
</tr>
<tr>
<td>171</td>
<td>M152</td>
<td>1337759503</td>
<td>1</td>
<td>HEIGHT</td>
<td>550</td>
</tr>
<tr>
<td>172</td>
<td>M676</td>
<td>1337759503</td>
<td>1</td>
<td>COLOR</td>
<td>BEIGE</td>
</tr>
</tbody>
</table>

Figure 5.1a: Randomly generated feature sets per entity in the database.

Figure 5.1 shows the generated feature set randomly generated by the feature generator. For example entity with entity_id M203 has the following feature set: the color of the ship is pink, the material of the ship is Foil and the height of the ship is 982 cm.
And this is how it looks like in our program:

These generated features for each vessel are also stored into the database. Then it looks like the data
in figure 5.1a.

Figure 5.2 shows the running example of the Entity Information Generator. The Screen on the right
side is HLA, displaying the simulation time and an acknowledgment of receiving AIS messages.
Currently there are 278 ships present in the simulation. The left side of the figure shows Entity
Information Generator performing a batch store operation. Moreover, it also shows acknowledgments that AIS messages are send to HLA.
Figure 5.3 shows an example of task distribution in our system. All tasks are distributed equally in each time step. This optimizes the system performance.
Figure 5.4 shows on the left side the Entity Information System, created lots of AIS messages 123 and AIS message 5 and acknowledgements that the generates AIS messages are stored in the database. The right part shows the acknowledgements of HLA that the generated AIS message for a specific vessel has been received.

Figure 5.5 shows an example of generated AIS message type 5. The figure shows four generates messages that were stored in the database.
Figure 5.6 shows an example of handling C2 request. Assume there are two entities, M1 and M2. C2 request AIS data for the time interval <39003500, 39003800> (is the time in seconds after 1970). But the database does not have these data, so it needs to be generated. The missing data is filled using recursion of the generating AIS task. If all requested data is generated, an acknowledgement will be send to C2.
5.2 Future work, improvements and recommendations

Although the Entity Information System has come a long way during this project, there is still a lot of space for improvement. As the history generation of AIS messages features have been partly discarded, we will not elaborate discuss the improvements for that. Instead, in this section we discuss the improvements on the Entity Information System, while making some recommendations on new generators and features as well.

Unfortunately, this project cannot cover all features that have been thought of, at the beginning of this project, like using different kind of sensors. TNO have made some choices to prioritize features. Some of them have made it to the milestone, while others have not. Live AIS generation and partly history AIS generation are two features that have been in the original requirements, but the history generation feature was not complete, due the missing generating algorithm, which cannot be finished in time.

The missing generating algorithm should be able to determine the positions, destinations and more static and dynamic AIS information of a vessel. This information can be used to determine the history of a vessel and filling in the missing data in the database, if C2 requests for it. We set up a framework which is able to determine the gaps in an interval and filling the gaps with task to completely fill in the requested interval. So the missing algorithm can be implemented in our system in the future within this framework, to improve the generating history feature.

Additionally, functions regarding to Daily Motion Patterns (DMPs) are also missing in the system, these features can be important to simulate a realistic environment for the sea traffic. The functionality of DMPs is part of SIMOBS, and unfortunately we have not used SIMOBS. The DMPs can be used to determine vessel behavior. Based on the behavior, analyzers can observe the vessels and determine whether a ship is acting suspiciously. The system can definitely be improved if these two features can be implemented. But for now it is impossible to develop these features due the end of our internship.

Future generators and features

While there were some generators concepts available when arriving at TNO, we have only implemented the AIS generator, feature generator and partly of the history generator. Due the limited development time, not all generators have been implemented. Generators like report generator or specific sensor generator can be easily integrated, as demonstrated by quick implementation of the feature generator. The report generator should be able to generate alpha report or INTELL report. We have managed the requirements for this generator in the Requirement Analysis Document, so the capability to develop this remains open for future work.

Also the thoughts of using specific sensors information like EO/IR or picture can be developed and applied to the current system. The requirements of these sensors can also be found in the Requirement Analysis Document, which can help with developing this feature in the future, if wanted. The same goes also for the features related to the use of Daily Motion Patterns.

We also recommend clustering entities per task in the future. This could optimize the number of task that is putted on the task list. In the current system each task is a standalone task of an entity, but
since it can occur that different entities have the same task to perform, it will be better to combine these tasks to one task, by clustering the entities. The Entity Information System is now able to store data in a batch, which means that at the end of each time step the system performs a store operation, containing all single store operations. For example, instead of performing 10 store operations the system performs one single store operation containing a conditions set of the 10 store operations.

Another recommendation involves test code. Although the system has several test classes it is still recommended to exhaustively test the current system with VR-Forces to check if the system is stable with about 2000 vessels. With stable we mean, if the system can handle 2000 entities at the same time without having to much delay. Additionally, more elaborated test classes for the system can be written in the future as well.

The last recommendation involves the “validity” of generated features. Currently, we use a set of features saved in the database to randomly generate a feature set for a vessel. It is recommended to check the database for the validity values of the features. More features can be easily added to the database manually.

5.3 Reflection and conclusion

Every project comes to an end and has a time of reflection. Where project members look back to the process of the project and reflect what they will have done differently. As expected this is also the case for our project. We have started the project in January, which was actually not our plan. We wanted to start doing a BSc. Project in September/October 2011, but due different kind of circumstances it has not worked out to start in that period. Before starting we had to do several kinds of practical matters, such as requesting a certificate of good behavior (VOG). This cost several weeks, which have lead to delaying the internship at TNO. So the first few weeks of January we have been working on the University, doing some literature study. But we did manage to have several interviews and meeting with our supervisor and other TNO employees to talk about the system and compose the requirements. We have spent a lot of time to finish the documents related to the orientation phase, it would be better if we had finished the documents before doing exams, so we could start designing the system a week earlier. This would have made it easier to have extra week for implementation and exhaustive testing.

One thing that we would have definitely done differently is meeting more often with our mentor from the university in the beginning of the project. To keep the mentor updated. At our second meeting our mentor could not remember that we have started with the internship at TNO. In the further progress of the project it did not seems necessarily to meet more often than we have done, since we also send a two weekly progress report every 2 or 3 weeks.

During the internship we worked together in the orientation and design phases. After finishing the design of the architecture we have divided the implementation of the system. One of us was focusing on the database side while the other one was focusing on the HLA part. This was a clever choice, recommended by our supervisor. If we had not split up the tasks, we could not probably have finished the prototype of the history generator as it is now. However, we do have moments where we have been doing pair programming. This has allowed us to get more knowledge about each
other’s code and helped us to improve the system. Since we do not always have the same idea how a function has to be implemented. Pair programming has allowed us to exchange and discuss the thoughts on how a function can be implemented most efficient. While we have worked separately sometimes, we both have been working on the documentations as well as the implementation.

We have tested the code manually as well as automatically for every milestone, but some test cases have not been needed any more for the next milestone, which have lead to a small amount of test cases for the final product. Next to that, we have used assertions in the code to do automatic checking, which have worked out pretty good. But in retrospect, we have tested less exhaustive in the final product than we have planned before, because the limited amount of time and the additional work and requests from other TNO employees.

The whole implementation was pretty tough if we look back to it. However the structure of the system seems to be relative easy to explain, but realizing it definitely requires a large amount of time and effort. The principle is clear receiving a callback, perform an action, store data and return the data. But it is more complicated, because we had to deal with time management, task management and data management. Some unexpected problems could occur, which have lead to frustrating moments. Some frustrating moments have occurred when we have encountered problems that could not be solved immediately and we could not find someone to help us with the problem. But most of the time the problems were solved the next day, when our brains are fresh and clear.

During the whole project we learned much about how a software product needs to be developed in a, for us unknown environment, where we are working with other employees. We do not have other students working on the same kind of project who can help us out, but we do have a supervisor who comes every week to us to discuss the progress and helps us with encountered problems and difficulties.

We now understand the importance of combining task management and time management, when the system continuously receives callbacks to perform an action. In addition, to the technical implementation difficulties, we also had to deal with some technical problems with combining the used software, for example linking libraries in Visual Studio, combining the HLA implementation with the database implementation. Furthermore, we have needed to find out how we can create an installer of our program, including all libraries and drivers that are needed to run our program.

These problems might sound easy solvable for some others, but due lack of experience it has required us much time to solve it. Normally these kind of difficulties can be solves by asking other students or mentors for help, but since we are only working with two of us, and none of us have much experiences with it. And the option of asking help has not always been possible, so we have been forced to find it out ourselves. This also has a good side, since we have learnt much from it.

Moreover, we have learned that documenting the development is also an important step in the development process, because the documentation can be used by other people to understand the behavior and design of the system, without having knowledge about the code itself.

Unfortunately, we have come in contact with the Daily Motion Patterns (DMPs). From the beginning of the project these patterns seem to play an important role. But in our final product we have not using any of the so called DMPs. Since the functionality of DMP are part of another component (Simobs), which we have not been using. Another point is developing a GUI for our program, we
would like to have developed a GUI for our system, but the point is that the GUI will not show some spectacular work. Since our program is running intern, it does not contain any graphics or interesting visualizations other than generated data messages and data.

Although, we may have underestimated the amount of work that needs to be done, including additional request from other TNO employees, and we have deviate a bit from our original plan, but we can say we are content with the resulting product. It might does not do anything visually for the user, but we have put a lot of effort in it to make it works as it should do. Our TNO supervisor is also satisfied with the results, since we delivered a fine architecture, which can easily be expanded with other generators. We have demonstrated this by quick implementation of the feature generator.

However, this is only a basic working prototype for generating AIS messages and features sets. By adding more generators to the system, we do believe that the Entity Information System can make the simulation more realistic and easier. Additionally, it can be very useful for MSA analysis (purposes).

In short, we have surely enjoyed the project/internship and we have learned much from it. The project not only contributes to our knowledge and development, but it has also improved our coding skills. But more important we have also been given the possibilities to experience how things are going on in the business world. We are glad that we could participate in the development of the Entity Information System, which is part the Sea Traffic Generator project.
References

<http://www.tno.nl/content.cfm?context=kennis&content=expertisegroep&laag1=3&item_id=28>

<http://www.gatherspace.com/static/agile_software_development.html>

<http://www.waterfall-model.com/>

<http://www.agilemodeling.com/essays/fdd.htm>

<http://tortoisesvn.net/>


<http://www.umlet.com/>

<http://www.mysql.com/downloads/connector/cpp/>


[10] "RESTful Web services: The basics", IBM. Visited on April 05, 2012  

# Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>AIS</td>
<td>Automatic Identification System</td>
</tr>
<tr>
<td>ADD</td>
<td>Architectural Design Document</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>C2</td>
<td>Command and Control</td>
</tr>
<tr>
<td>DMP</td>
<td>Daily Motion Pattern</td>
</tr>
<tr>
<td>DUT</td>
<td>Delft University of Technology</td>
</tr>
<tr>
<td>Eoil</td>
<td>Electro Optic Infra Led</td>
</tr>
<tr>
<td>FDD</td>
<td>Feature Driven Development</td>
</tr>
<tr>
<td>FOM</td>
<td>Federation Object Model</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HLA-TM</td>
<td>High Level Architecture Time Management</td>
</tr>
<tr>
<td>HLA</td>
<td>High Level Architecture</td>
</tr>
<tr>
<td>HMI</td>
<td>Human-Machine Interaction</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent Tutoring System</td>
</tr>
<tr>
<td>KML</td>
<td>Keyhole Markup Language</td>
</tr>
<tr>
<td>LBTS</td>
<td>Lowest Boundary Time Stamp</td>
</tr>
<tr>
<td>M&amp;S</td>
<td>Modelling and Simulation</td>
</tr>
<tr>
<td>MSA</td>
<td>Maritime Situation Awareness</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>NOSQL</td>
<td>Not only SQL</td>
</tr>
<tr>
<td>OOP</td>
<td>Object Orientated Programming</td>
</tr>
<tr>
<td>RAD</td>
<td>Requirement Analysis Document</td>
</tr>
<tr>
<td>RDS</td>
<td>Remote Desktop Server</td>
</tr>
<tr>
<td>SBM</td>
<td>System &amp; Behavior Modelling</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SEM</td>
<td>Synthetic Environment Modelling</td>
</tr>
<tr>
<td>SSE</td>
<td>Simulation Systems Engineering</td>
</tr>
<tr>
<td>SIG</td>
<td>Software Improvement Group</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Orientated Architecture</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>STG</td>
<td>Sea Traffic Generator</td>
</tr>
<tr>
<td>TSO</td>
<td>Time Stamped Object</td>
</tr>
<tr>
<td>VR</td>
<td>Virtual Reality</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Appendices
Appendix A: Sea Traffic generator description (in Dutch Only)

Bachelorproject
Sea traffic generator

TNO Defensie en Veiligheid is één van de vijf kerngebieden van TNO en levert vernieuwende oplossingen om de algehele veiligheid van de samenleving te bevorderen. De opgebouwde kennis passen we in de praktijk toe voor en met onze opdrachtgevers, zowel voor militaire als civiele doeleinden.

TNO Modeling, Simulation and Gaming

De opdracht
Binnen de afdeling beschikken we over een aantal verschillende software tools, die gezamenlijk een goede basis vormen voor het ondersteunen van militaire experimenten en oefeningen met behulp van simulaties. Een veelgebruikte tool is een scenario generator. Deze applicatie stelt een gebruiker in staat om een gsimuleerde omgeving te "vullen" met objecten (personen, voertuigen, schepen, vliegtuigen, etc.) waarmee een experiment wordt uitgevoerd. Gsimuleerde omgeving zijn echter vaak erg 'leeg' wat betreft aanwezige objecten die niet direct onderdeel zijn van het scenario, zoals omstanders en voorbijgangers. Dit is een gemis omdat de aanwezigheid van deze "normaal" gedragende objecten een cruciale rol speelt in werkelijke operaties. Dat geldt uiteraard voor de civiele wereld, maar zeker ook voor het moderne militaire optreden.

Hoewel de simulatie van deze objecten niet altijd bijster complex hoeft te zijn (het moet immers vooral realistisch ogen) kost het op dit moment nog veel tijd om een gsimuleerde omgeving hiervan te voorzien. De reden hiervoor is dat deze veelal handmatig moet worden toegevoegd. Het ontwikkelen van een scenario waarin doelstellingen gerealiseerd kunnen worden is voor een scenario ontwikkelaar al een zeer arbeidsintensief proces. Hierdoor komt het verrijken van een scenario met "normale" objecten vaak niet meer aan bod, wat weer ten koste gaat van het realisme.

Het doel van deze stageopdracht is het realiseren van een "automatische" scenario generator op basis van Daily Motion Patterns (DMP). Middels het definieren van wonen en werk gebieden en DMP's door de scenario ontwikkelaar dienen meerdere objecten gegenereerd te worden als onderdeel van het scenario (b.v. naar werk/huis gaan). Specifiek voor deze opdracht willen we binnen de maritieme context scheepssverkeer en vis gebieden "bevinden".

De lengte van dit project bedraagt 11 weken. Het is mogelijk deze opdracht door een team van twee tot vier studenten te laten uitvoeren. TNO biedt een maandelijkse vergoeding.

Geïnteresseerd?
Ben je enthousiast over de hierboven beschreven opdracht en heb je interesse? Neem dan contact op met:

Dhr. R. Krijnen
tel: (088) 86 63969
e-mail: robbert.krijnen@tno.nl

of

Dr. ir. Gerwin de Haan
tel: 015 27 81 445
e-mail: g.dehaan@tudelft.nl
Appendix B: Orientation Report

Orientation Report
# Table of Contents

1. Introduction .................................................................................................................. 68

2. Sea Traffic Generator .................................................................................................. 70
   2.1 Goal .......................................................................................................................... 70
   2.2 Scenarios .................................................................................................................. 70

3. Research Analysis ......................................................................................................... 71
   3.1 AI Approach .............................................................................................................. 71
   3.2 Environment Generator ............................................................................................ 72
   3.3 Training Simulation ................................................................................................. 72
   3.4 Smart objects ............................................................................................................ 73
   3.5 Pluggable brains ....................................................................................................... 74
   3.6 COVE ITS ................................................................................................................. 75

4. System ............................................................................................................................ 76
   4.1 Maritime Situation Awareness .................................................................................. 76
      4.1.1 Daily Motion patterns ......................................................................................... 76
      4.1.2 VR-Forces ........................................................................................................... 77
      4.1.3 High level architecture ...................................................................................... 77
      4.1.4 Automatic Identification System ....................................................................... 78
   4.2 Comparable systems ................................................................................................ 79
      4.2.1 ATC .................................................................................................................... 79
      4.2.2 Di Guy ............................................................................................................... 80
      4.2.3 SIMOBS ............................................................................................................ 81
   4.3 Consideration ............................................................................................................ 81

5. Problems ......................................................................................................................... 83
   5.1 Common problems ..................................................................................................... 83
5.2 Technical problems .................................................................................................................. 83
5.3 Functional problem .................................................................................................................. 83

6. Tools ........................................................................................................................................ 84
   6.1 Visual studio 2008 ................................................................................................................. 84
   6.2 VR Forces 4.0.2 ..................................................................................................................... 84
   6.3 Dropbox and SVN .................................................................................................................. 84

7. Implementations ....................................................................................................................... 85
   7.1 Service Oriented Architecture .............................................................................................. 85
      7.1.1 ReST ............................................................................................................................. 86
   7.2 Algorithms ............................................................................................................................ 86
   7.3 Generating history .................................................................................................................. 87
   7.4 Database ............................................................................................................................... 87

8. Conclusion ................................................................................................................................ 88

Glossary ........................................................................................................................................... 89

References ...................................................................................................................................... 90
1. Introduction

The shipping industry plays an important role in the world. It is more than transporting goods by sea. Daily thousands of vessels are sailing around the world, not only to transport some goods from point A to point B. Some of them are doing business like fishing, others are sailing for fun. And some might be committing a crime, like illegal immigration, drugs trafficking or piracy. Because of the large amount of vessels on the sea, it is hard to identify and check every single ship, whether they are doing their daily job or hiding something for the marine observation system. It would be a lot easier if vessels can be tracked and their behavior analyzed automatically by different type of sensors. But to determine whether the sensors are useful and effective, it is important to test it first in a simulation before experimenting it in the real world. In order to simulate the ‘content’ ships with various behaviors are required, among others. This condition leads to the need of developing a Sea Traffic Generator.

The main goal of the Sea Traffic Generator is: “Generate a realistic representation of sea traffic and compute/store the history of the vessels according to their behavior”. A detailed description about the project assignment can be found in chapter 2.

Generating a realistic worldview of the sea traffic is pretty tough, because you have to deal with several aspects like the environment and the way objects behave. Questions like: "how should the environment look like?", "What objects should be present?", "In which way should the object perform an action?", "What is the behavior of the objects?" play an important role during a simulation. To understand more about these aspects, we analyzed some papers, which are detailed in chapter 3.

The idea to develop the Sea Traffic Generator results from a previously developed system called SIMOBS. This system is an application which you can use to simulate an environment with villagers having specific behavior, also known as a Pattern of Life. For example, a villager is going to work in the morning at eight o’clock and come back home at six o’clock. The SIMOBS principle can also be used for ships. Each vessel has its own behavior. For example a fish boat leaves the harbor in the early morning, to sail to a fishing area to catch some fish. This pattern is a specified Daily Motion Pattern (DMP) of a fish boat. In general, Daily Motion Patterns contain behavior information about a certain object. In our project these objects are vessels. If one day a vessel differs from its “normal behavior” it would be very suspicious. For example a fishing boat sails from a harbor to a suspicious non-fishing area. The system that analyses the sea traffic might think that this ship is performing something suspicious, because the ship differs from its normal behavior. Maritime Situation Awareness (MSA) is such a system that analyses suspicious behavior of the sea traffic. More about the DMPs and MSA can be found in chapter 4.

Moreover, we have searched for comparable systems to the one we want to develop. By analyzing and comparing these systems we can create a consideration with advantages and disadvantages. Based on this consideration we can improve the development of Sea Traffic Generator.

According to the previous chapters we will take a look at the problems that can occur, when developing the Sea Traffic Generator. In chapter 6 some tools will be introduced which will be used during this project. Chapter 7 will describe some details about the way we will implement the system.
We also explain what kind of algorithms we will use and how the Daily Motion Patterns play an important role in the implementation. At last but not least a conclusion will be taken based on the chapters before.
2. Sea Traffic Generator

Marinetraffic [18] is a system that provides real-time sea traffic on a map. When the system is running, you can see the different kind of vessels, and their behavior on the sea. These data are synchronized with the behavior of the vessels which are currently “operating” in a specific area.

We want to create a simulation system where we can generate sea traffic. This idea is comparable to the above mentioned marinetraffic. Generating a realistic environment takes a lot of time, so we want to develop a system that is able to automatically generate an environment close to the real world. But if we want to develop a system like marinetraffic, we need to find a way to generate the behavior of vessels, so the simulation looks as real as possible. It is not just about the behavior of a vessel from the point we start the simulation. It should also be possible to generate information about a vessel before we start running the simulation. When the simulation starts all vessels should have information about their behavior in the past, like what time and where they have sailed to.

This chapter will describe some details about the Sea traffic generator project. What the purpose of this system is and what kind of scenarios can occur. The original project description can be found in appendix A, on page 29.

2.1 Goal

The goal of this project is to generate a realistic world view in a particular situation, with regard to sea traffic. The ships have to behave realistically and it should also be possible to trace the history of a vessel, so even from a random starting time a realistic initial scenario is always present. Generating the behavior and the history will be made possible by using Daily Motion Patterns.

2.2 Scenarios

During the simulation different kinds of scenarios can occur. We will describe the basic scenarios we would like to have in our system.

1. All vessels are sailing according to pattern which is assigned to them at start of the simulation. Deviant behavior does not occur/ the user perceives nothing suspicious.
2. All vessels are sailing according to pattern which is assigned to them at start of the simulation. But suddenly deviant behavior occurs/ the user perceives suspicious behavior.
3. A role player intervenes in scenario 1, by changing some pattern of vessels. On this way the vessels behavior will differ from the original pattern. Deviant behavior occurs.

These are the three basic scenarios that can occur during the simulation of the Sea Traffic Generator. Of course more scenarios can occur, but these are the basic ones. All other scenarios are a variation of one of the scenarios mentioned above.
3. Research Analysis

In this chapter we will analyze some papers which we used in our research. These papers provide more information about subjects that play a role during a simulation. We will give a short summary about each paper, followed by points that can contribute to our system.

3.1 AI Approach

Paper:
*A Game AI Approach to Autonomous Control of Virtual Characters [11]*
*By Dill et al.*

The paper mentions why there is a strong need for developing Artificial Intelligence (AI) for virtual characters. Moreover, the paper explains the AI architecture of a game called Angry Mother, and compares and contrasts it to relevant game AI approaches.

By developing AI for virtual characters would bring the following benefits: First, training cost can be reduced, because the characters are virtual, there no need to pay for human role-players. Second, accessibility increases. Third, consistency can be improved.

The following characters need Artificial intelligence to function properly:
- Autonome – are able to function by them self with less or non-human input
- Reactive – can adapt themselves if the scenario or the actions of the trainees are changed
- Nondererministic – The viewer should not be able to see things twice
- Culturally authentic – functions like a person of a certain would act
- Believable – Acts on the way a real human would act

For the decision making a Component Reasoner can be used, which is similar to the Behavior tree (BT) architecture. A Behavior Tree contains a huge amount of selectors. Every time the character should choose an option until it arrives at a concrete option. The Behavior Tree relies on simple Boolean operations for their selectors.

Behavior Tree has 2 major advantages: first, the decision approach avoids spending processing time on irrelevant decisions, because the AI is structured on a natural way such that independent decisions are separated. Second, the options in the Behavior Tree can appear more than once in the three. On this way the functionality does not have to be re-implemented every time that it is used. Applying this approach, the most appropriate approach for each decision can be selected.

Component Reasoner uses the principle of BT, but instead of using selectors it uses reasoners. Reasoners are configurable via XML and can be more complex than selectors. The advantage of using a component reasoned is that for each decision the most appropriated approach can be selected.

Tracking
- Using degree of freedom (DOF) system, for example use camera to track position and orientation of the head. DOF would also be used in our case with Daily Motion Patterns
- RFDI (Radio-Frequency Identification) system. In our system we use AIS.
Interesting points:
- Using XML files, where configurations en entities can be modify. In this paper they use a data-driven approach to accelerate the configuration.
- In the paper a Component Reasoner is used to make a decision. In our case it is up to role player to decide what to do. The role player can intervene at any moment.
- The way how tracking an entity can take place. In the paper they use a RFDI. While in our system we use AIS and different kind of algorithms to calculate the location of an object.
- All vessels in the Sea Traffic Generator have their own pattern to perform their “job”. Each pattern describes the behavior of a vessel that is coupled to the pattern. These patterns can be seen as AI for the ships. Because of these patterns, no human role players are needed to control these vessels.

3.2 Environment Generator

Paper:
An instructor-centered Declarative Approach to Modeling Synthetic Environments [17]
By Kuijper et al.

The basic idea of this paper is how to develop a scenario generator, which provides a scenario maker to create a scenario from scratch to a sketch. This sketch will/could be elaborated to a complete scenario. A wide range of objects (like bushes, roads, cities, rivers, seas) can be placed in this scenario.

This is very useful to us, because the Sea Traffic Generator is required to generate a virtual representation of the actual, real-life situation. This representation combined with a good performance could become very handy. Here the procedural operations are between the 100milliseconds and three seconds. Only continuously modifications to the 3D geometry in large landscapes could lead to a bottleneck.

The medium size of the landscapes is about 1000 km², large enough for what is expected from us related to the size of the scenario. There is also the possibility to generate a scenario based on a GIS-import (geographical data). One of the possibilities in the Sea Traffic Generator is to load actual geometric data, so that this feature is useful.

3.3 Training Simulation

Paper:
Understanding the Impact of Intelligent Tutoring Agents on Real-Time Training Simulations [15]
By Brawner et al.

The paper is about which role agents play on a real-time training simulation. The agents can provide feedback within a scenario, so the trainee can pick up these hints, before making any decisions.

Comprehensive training model:
- Can understand an instruction much better, even if the scenario changes.
- Is able to predict the learner’s response and the action that is taken.
- Give feedback to the learner based on the decision which is made.

In our case, the vessels can be seen as agents. They use the Daily Motion Pattern to obtain information like on time x ship y should be on place z. More information about an entity can be
received from sensors. But the difference with the paper is that in our case the agents are not tutoring learners. Each vessel can be controlled by a role player who can create events manually, with DMP or with VR forces.

### 3.4 Smart objects

**Paper**

*SmartObjects and Agents for autonomous high population simulated environments* [5]
*By Blank et al.*

This paper describes that by use of the HFSM (High Finite Turing Machine) Minds, a strong object-oriented design with polymorphism and Smart Object leads to a large reduction in costs, which occur in a simulation containing multiple entities.

**Reduce costs**

This technique can be very useful in our Sea Traffic Generator project, since we also have to deal with a large amount of entities. The estimate costs of the Pattern of Life, which is comparable with our Daily Motion Pattern, will be from

\[
\text{Reduced to:} \quad N \cdot P + D + N \cdot (D!)
\]

With:

- \(N\) = number of virtual characters
- \(P\) = actionable props
- \(D\) = destinations of interest
- \(C\) = number of distinct crowds (significantly less than \(N\))
- \(A\) = number of crowds and destinations of interest simultaneously

Using Smart Object provides us a gain of performance.

**Path planning**

They further use the popular A* search algorithm [], to calculate the routes. Mention that these routes are the cheapest paths, not necessarily the shortest. Here, they used it sidewalks, crossing places. Probably, we will use the path planning to create the shipping lanes and perhaps high risk areas, in relation to piracy.

**Intersection testing**

In order to avoid collision, they use fast collision detection algorithms. We should also avoid collision between separate boats and between the coast and the water. To realize it, we could also use collision detection algorithms, to increase performance and reliability.

**Communication**

A communication protocol is used for the agents to communicate among themselves. We should also use a possibility like that for our DMPs, or just all entities, to avoid collision between entities.

**Scheduling**

Modify the patterns, which are used here, with a certain variation, to increase the level of realism. To realize this, they make use of certain methods, which per (sub)set of the crowd, indicate their goal. Also the time to reach this goal or changing the (HFSM) state and the estimated time, before the
action will be executed by the system are options for certain sets containing crowd. We could also use methods like this to achieve variations in our DMPs, to improve reality.

**Possible problems**
When you create an extended scenario, like 100 entities, 100 paths should be calculated. This results in a loss of performance, especially using C++. “Let the agents partly following other agents”, could be a proper solutions in this case. Probably, another solution should be found in our case.

### 3.5 Pluggable brains

**Paper:**
*Pluggable brains: Separating the Intelligent Behavior from the Simulated Player [16]*
*By Kerbusch et al.*

The paper describes an approach of separating behavior component from the simulation engine. AI is an important component which is often used in a simulated environment. For example, AI can be coupled to virtual role-players.

In the last few years some trends have been developed that changes the way AI is used. One of the trends is the use of commercial-off-the-shelf (COTS) available software and tools. The other trend is the human-machine interaction (HMI) research with respect to autonomous systems. The Modeling and Simulation (M&S) adopted these trends and developed the brains (AI) pluggable. In this approach the integrated AI tools and software components are decoupled from the simulation engine.

The pluggable brain approach needs a communication platform to exchange information and data with other component. In this paper the authors chose for HLA (High-Level-Architecture).

The reasons for using HLA are:

- HLA is a shared data-exchange mechanism.
- It is widely accepted; many COTS simulations support High Level Architecture.
- Lots of HLA supporting tools are available.
- Newest version of HLA makes use of FOM (Federation Object Model) modules. FOMS ensures that information exchanged on HLA will be described unambiguously.

**Decoupling between brains and simulation**

1. Decouple the AI brains from the system, which means placing the brains outside the simulation. On this way the develop can chose its own AI middle ware.
2. Reusing shared concept between brains and simulation.

**Benefits of decoupling:**
- It enables researches to experiment with various man-machine collaboration concepts.
- Components can be reused.

**Consequence:**
- Risk in data-exchange agreements getting out of sync or overlapping, when large number of federates participated in a system.
By analyzing this paper, the use of HLA clearly described. In our system we also use High Level Architecture as a communication medium between all of the components of the system. With this paper we understand more about how the possibilities of HLA.

### 3.6 COVE ITS

**Paper:**

*Automated support for learning in simulation Intelligent of shiphandling [21]*

*By Peter et al.*

Paper mentioned how to train students with an artificial intelligence tutoring system (ITS). Using such kind of systems can reduce manpower cost and enables a reduction in dollar. The US navy set up a shiphandling simulation, named COVE (conning officer virtual environment) ITS. In this simulation junior conning officers can be taught the basic shiphandling courses. One of the main goals of the system is to train conning officers to take the right decision on the right time, when they see some useful visual cues where the ship is. To make the correct decision on the right time, ship control skills and observing skills are very import.

Relevant for our work:

- Can be used understanding how shiphandling works.
  
  Response of a ship is determined by hydrodynamic and other physical forces. It also depends on characteristics of specific ship types, like what the effect would be of combinations of rudder, engine, and tug orders in real-time.

- Which factors are related to shiphandling;
  
  Environment, variation of variables of a ship, like rudder, engine, and tug.

- How to deal with observed data, what actions need to be taken.
4. System

This chapter provides information about the system we want to develop. First we will take a look at the Maritime Situation Awareness, and given some details about the components that play an important role in Maritime Situation awareness. You can think of the use of patterns, the architecture and the sensors to perceive information. Secondly we will describe some existing systems that are similar to the system we want to develop. And at least a consideration of these systems will be made, according to their advantages and disadvantages.

4.1 Maritime Situation Awareness

Maritime Situational Awareness (MSA) is to obtain a common view of a situation of the maritime environment to operate properly and effectively. By observing ships that sail on the sea, based on their Daily Motion Patterns, a MSA system can analyze the behavior of these ships, to detect suspicious behavior.

Changes in the international security situation led to a shift in military operations. Besides operating in the higher spectrum of violence, the importance of crisis management operations has increased. This shift has led to the introduction of the concept of "Maritime Security Operations." Here are issues such as counter-terrorism, piracy and armed drug trafficking purposes can be understood. [29]

4.1.1 Daily Motion patterns

Daily Motion Patterns, abbreviated to DMPS, are be used to describe the behavior of simulated objects as natural as possible. So eventually a complete visualization gives a realistic scenario. In our case, each vessel type will have its own Daily motion pattern. Moreover, each vessel can vary within their own DMPs, so each vessel can have their own “unique” shipping route. Figure 1 shows an example of a DMP from a vessel.

Figure 1. Example of a Daily Motion Pattern
DMPs can also be described as 3DTI:

- **3D position and place (location)**
  
  *Absolute place/position against others.*

- **Time / history**

- **Interaction:** interaction between different entities

- **Intelligence/information:** the knowledge we about the entities

In the Sea Traffic Generator we will use Daily Motion Patterns to describe the behavior of each ship. All vessels during the simulation will act according to the Daily Motion Pattern which is attached to them.

### 4.1.2 VR-Forces

VR-Forces is commonly referred as a Computer Generated Forces (CGF) tool. It is a powerful and flexible simulation environment for generating and executing battlefield scenarios. VR-Forces can be divided into two parts; the GUI and the SIM Engine. [9]

First, the SIM Engine is responsible for creating/generating object and events. More specific, the entities will follow roads or routes and will interact with other entities or the environment. Also the detection of suspicious behavior and avoiding obstructions are possibilities the engine provide.

VR Forces GUI, as second, is an editor which allows building scenarios by positioning entities, creating routes/waypoints and assigning task to the entities. By using the 2D map, a scenario can be made in a very short time. The GUI also allows the user to see the scenario in 3D. The geographic landscape can also be imported using GIS-Data or 3D data, created by a 3D modeling tool.

VR Forces also provides the possibility to manipulate their software architecture, using plug-ins to modify or extend the delivered functionality. [26]

In the Sea Traffic Generator we will use VR-Forces to generate the sea traffic objects and events. The VR-Forces GUI will be use to create the graphical user interface.

### 4.1.3 High level architecture

We also mention this with the abbreviation HLA. HLA is the widely used standard for coupling distributed simulators. It can be seen as a communication medium, which exchange object and events. For example VR Forces creates objects with the following characteristic properties: Ferry, color white, owner John. This information will be published on HLA. Other pluggable components who are interested in this information can subscribe to HLA, and gained the published information. The benefit of this is that not every component will be overloaded with the information published on HLA.
4.1.4 Automatic Identification System

This is also known as AIS. This system is used for identification of vessels in the sea traffic. It sends signals to other vessels and instances of the harbor. Such signals contain information like identification number, name, length, width, ship type, current position, etc. These transactions lead to a more safety sea traffic.

The advantage of AIS is, not every ship has an AIS. The signal is limited and the identification information mentioned in the AIS are not filled in properly. This leads to losing information, so this system is used in the sea traffic.

- **RADAR**

Rader is sensor which perceives information about Rader Cross section, abbreviated as RCS. This type of sensor indicates with Radar Cross Section-value [28] whether an object is detectable by radar. A higher RCS-value means that an object is easier to detect. The degree of reflection, i.e. level of detectability, is dependent on the material, size, and the reflection angle of the object.

- **Picture**

This sensor provides information about a vessel based on a picture. All signature information about a ship can be requested. *For example the color, length or width of a vessel.*

- **INTEL**

This is a kind of secret agent, which indicates for example when a smuggling boat in a certain location at a certain time is signaled. Using the name and associated MMSI-number the radar cross section of a ship can be derived by checking the ship register. Then *Persistent surveillance* can be applied on this ship, by collecting as much history about that ship as possible.

- **EO / IR**

It stands for Electro Optical and InfraRed. This type of sensor gives information from land, air and satellite. You can think of satellite images from an area where a vessel is present.
4.2 Comparable systems

This part will describe some systems which are comparable with the proposed system. Some of them use the same techniques as the Sea traffic generator. Others have the same idea, i.e. to analyze suspicious behavior.

4.2.1 ATC

Agent Technology Centre (ATC) developed a system where they distinguish behavior models in three different kinds of vessel (types). The first one are the merchant vessels (figure 2a), the second one (figure 2b) are the pirate vessels, ending with Navy-vessels (figure 2c). [1]

The shipping routes are generated by the Shortest-path-algorithm, based on risk assessment. In the way to the chance to encounter the pirates is minimized.

By using a finite state machine (FSM) the behavior of the ship (agent) can be described, to simply the implementation. In figure a finite state machine is shown.
Google Earth is used for the visualization, in combination with the JAVA API KML library. This library is used to describe and visualize geographic data, based on the XML language schema. [12]

They use the A-lite platform as a communication medium to ensure the interactions between the agents. A-lite is a software toolkit that helps with some implementation steps during construction of multi-agent systems and multi-agent simulation. [22]

4.2.2 Di Guy

Guy Di has a whole package of visualizations and behaviors varying from the army to overcrowded cities. [10]

Features
The SDK in C++ provides support for multiple platforms. It claims to have a powerful API, a high-performance motion engine, many different styles, movements and behavior.

AI
- Measurements were crowds with the click of a mouse
- From self-navigating, (very) smart characters
- Pattern of Life
- Building and land associated with those entities
- Populate a field with a distributed simulation

Scenario
- Take the scenario in a 2D-3D environment, this speeds up the process
- Place and government characters, real-time
- Possibility to all Di-Guy SDK entities with a mouse to place
- Ability to add special effects (fire, smoke, spores, and the weather)
4.2.3 SIMOBS

SIMOBS, developed by TNO, is a system that realizes an independent representative behavior generator. The simulator is mainly developed to inject suspicious behavior in a relevant scenario among normal behaviour. In order to realize simulation of behavior, activities and interaction between objects, SIMOBS makes use of Artificial Intelligence. [25]

Daily Motion Patterns where used to generate behavior in the scenario. They link a unique DMP, by varying within the pattern, to every unique object, to provide a realistic output. By using FSM, written in XML, it is very easy to modify both the scenario and the DMPs, to improve the usability of the user.

With SIMOBS, the scenario developer is possible to create a scenario, which can be saved and loaded, from scratch in a GUI, called VR Forces GUI. It is also possible to import GIS-data to this module, which they use to create the scenario and simulate it, to obtain a realistic representation. By using the VR Forces module, algorithms like path planning and collision detection are included in its simulator.

A nice feature is dead reckoning, which improves the performance by only sending the initial information including its destination, from which the actual location can be calculated, instead of providing a real-time position update. The benefit of this is that use the bandwidth can be reduced in the communication of positions.

4.3 Consideration

This part will describe the disadvantages and advantages of each system, with regards to the system we want to implement.

<table>
<thead>
<tr>
<th></th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| ATC AgentC | - Shortest path algorithm based on risk assessment.  
- Use the Final State Machine (to describe behavior) in order simply the implementation.  
- Using Google Earth as visualization.  
- Use A-lite platform as communication medium as platform between agents.  
- Can use GIS- and vessel data provider. | - Limited to real-time scenarios only, based on GIS Data. It is not possible to create sketch-based scenarios.  
- Avoid interesting areas with shortest path algorithm. We would have the opportunity to analyze risky areas.  
- Possibility to decrease performance, caused by communication redundancy. |
| Di Guy | - Using Pattern of Life  
- Self-navigating smart characters.  
- Possibility to create sketch-based scenarios.  
- Easy measurements with mouse-clicks.  
- Convenient populating scenario.  
- Can be visualized in 2D as well as in 3D.  
- Possibility to control the entities, real-time. | - It could take a long time to create/generate realistic scenarios from scratch.  
- Requires a huge amount of time to implement/develop.  
- The behavior in Di Guy is based on human acting/behavior, using other entities like vessels, will not give a realistic view/simulation. |
| SIMOBS | - Injection of suspicious behavior.  
- Using AI techniques to generate behavior, activities, and interactions | - DMPs cannot be modified by the GUI.  
- It is not possible yet to couple the B-HAVE module. |
between entities.
- Using DMPs patterns.
- DMPs can be varied for every single entity.
- DMP modeled as FSM, in XML. So the user can easy to modify en extend.
- Moreover, user can add new areas and objects in XML.

- SIMOBS cannot be decoupled from VR-Forces.
- Limited amount of different DMP types.

All above mentioned systems have its own advantages and disadvantages, but we will try to develop a system that will cover most of the requirements according to the client wishes. We will also try to cover most of the disadvantages of the above mentioned system. But since our main task is to develop a system which is able to generate sea traffic history, some components, like “graphical user interface”, “Daily Motion Pattern” will not be developed by us.

ATC Agency does not meet the requirements of our system, so this cannot be used to develop the Sea Traffic generator. First of all it is not possible to create sketch based scenarios, which is a must the Sea Traffic Generator. Secondly the shortest path algorithm is not the algorithm that we would like to use, because within the Sea Traffic Generator all behavior of vessels is based on Daily Motion Patterns.

Di Guy is a better approach for our system, but it also has some limited features like the behavior in Di Guy is based on human acting, and even if the behavior change to the behavior of vessels, it will not give a realistic simulation of the sea traffic. But we can use Di Guy to observe how they deal with Patterns of Life and analyze the algorithm they used for generating the sketch based scenario.

SIMOBS is the system that will be extended and adapted such that this is applicable to sea traffic. It will be used to generate a sketch based environment, generated behavior, and activities, based on Daily Motion Patterns.
5. Problems

In this section we describe the problems which can occur during the development of the Sea Traffic Generator. These problems can be divided into three categories. The first one contains common problems, the second part contains technical problems, and the last part contains functional problems. Keeping in mind that our main goal is to develop a system, which is able to generate sea traffic. With the most important goal: “generating vessels history based on Daily Motion Pattern”.

Since we do not develop the whole system, we depend on the work made by other employees of the internship. We do not understand exactly how every single component work in details; we just did some research to the basic working principles of each component we will use. Because it would take a lot of time to understand all components in details. This is also not expected from us.

5.1 Common problems
- Not every ship has AIS
- How to deal with vessels, outside the observed area.
- Dealing with lots of information obtained from interviews with clients/employees.
- Limited documentation about the components used in the Sea Traffic Generator.

5.2 Technical problems
- Restricted knowledge/experience of the C++ and libraries/plug-ins.
- Knowledge about requesting database information using HLA environment.
- How to store lots of data at the same time, continuously.
- Creating a new graphical user interface (GUI) in the existing VR-Forces GUI.

5.3 Functional problem
- The way to the convert the observed information of sensors to useful information.
- Communication between all components couples to HLA.
- The way to generate history according to Daily Motion Patterns.
6. Tools

This chapter provides information about the tools that we will use during this project. We did not have to search for alternative tools, because the tools described below are used by TNO.

6.1 Visual studio 2008

Visual studio 2008 is an integrated development environment (IDE) from Microsoft. It contains the whole developing package: GUI-development, Object-oriented programming support, Debugging/Testing. It is also possible to use built-in plug-ins to create classes, database schemes and more like that. VS 2008 supports all different C-type code languages: C, C#, C++. Project management is also a powerful option of this environment, very useful to combine with the usage of a subreversioning program like SVN.

6.2 VR Forces 4.0.2

VR Forces is a C++ toolkit to generate and simulate a realistic scenario. It contains a both a GUI and an ENGINE.

The GUI contains extended features in creating the environment. It allows you to build the scenarios by positioning (in our case) boats, create routes and assigning tasks to objects like the boats.

The ENGINE provides the real-time simulation. VR Forces is able to visualize the simulation both in two- and three-dimensional.

6.3 Dropbox and SVN

Dropbox is a Web-based file hosting service operated by Dropbox, Inc. that uses cloud storage. Which allows users to share and store files. (Wikipedia, Dropbox Service). Dropbox users can access their files via internet or installing the Dropbox app for mobile cell phone or computer. One can restore a file when it is deleted or restore a previous version of the file. Using Dropbox allows users to exchange files easier, for example a user can synchronize some files between his desktop and laptop easily by uploading it via Dropbox, It not necessary to email to yourself anymore. We used Dropbox in the first few week to exchange documents with each other. On the moment when we started working at TNO, we changed to SVN because of security and confidential reasons.

SVN is source control software for Microsoft Windows, which has the same principle of Dropbox. The main difference is that within SVN it is allowed to work with more people on the same file, and within Dropbox you will always receive a conflicted file. When more people are working on the same file on SVN, SVN will merge the different changes.
7. **Implementations**

To have a good start with the implementation phase, we search for some implementation possibilities that deal with our system. We did research to the architecture of the system we wanted to design and searched for some algorithms which we could use.

7.1 **Service Oriented Architecture**

Abbreviated as SOA. The principle of SOA is reusing components/service. So that these components/services can be adapted easily, without spending lot of time to re-implement the same functionality of a component. Figure 4 shows the basic SOA principle. While Figure 5 shows a more detailed example of a Service Orientated Architecture.

SOA consisting of various "IT components" that can be reused, like:
- Business concept
- Managed by the organization, practices and standards
- Designed for more flexible enterprise software implementation and elasticity

![Figure 4. The basic principle of SOA.](image)

![Figure 5. An example of detailed SOA. (By Wikipedia)](image)

We will use a Service Orientated Architecture to communicate between two actors. Which are the role player and the user behind the application table.
7.1.1 ReST

ReST stands for Representational State Transfer and is introduced by Roy Fielding [13]. ReST is useful to implement Service Orientated Architecture component and easy to understand. It allows to works with HTTP operators like: GET, POST, PUT and DELETE, which can be used for query handling in our project. Namely, these operators can be used to SELECT, INSERT, UPDATE and DELETE from database tables. [20]

Using for example this request object [24]

```plaintext
GET /customers/1234
Host: example.com
Accept: application/vnd.mycompany.customer+xml

<customer>...</customer>
```

Could be processed to this database query of this type:
SELECT * FROM customers WHERE customers_id = 1234
and returning an XML-object.

If we want to request data from a database through a network using Service Orientated Architecture, ReST will become very useful to us.

7.2 Algorithms

Because we can reuse the B-HAVE component implemented in VR-Forces, we do not need to implement any new algorithms for subjects as path planning or collision detection. The B-HAVE plug-in uses the dead reckoning principle (described in SIMOBS). [4]

But we searched for an option algorithm which can be used in the future.

Path planning
To realize the path planning for our shipping lanes, we should possibly use this algorithm. The A* search algorithm is an extension of the former Dijkstra algorithm. Using heuristics, it provides a better performance.

By using a distance-plus-cost heuristic, summing the path costing function and the admissible heuristic estimate, to determine the order in which the search visits nodes in the tree. The admissible heuristic stands for the shortest optimal way to reach the goal (node).

Collision detection
Since the collision avoiding algorithm is already implemented in the B-HAVE plug-in, it is not necessarily needed to describe one or multiple kinds of algorithms to detect and avoid collision between entities or between entities and the environment.
7.3 Generating history

One of the goals of our project is to generate history of vessels. The history is actually the behavior of a ship. If we start the simulation and some information is missing, then the missing information should be generated.

History of a vessel can be generated on two ways:

Using Daily Motion Patterns

The behavior of a vessel is based on a Daily Motion Pattern. So using these patterns is should be able to compute the behavior of a ship in the past and in the future.

During a simulation the history of a vessel can be stored, by saving the time and location once in a while. Based on these storing points a history of a vessel is generated.

Another way to compute the history is to use the backtracking principle. For example a cargo sails with a certain speed toward the harbor of Rotterdam. Based on the speed of this cargo, it is possible to track back where the cargo was an hour ago, or even few days ago.

Influence of a role player

A role player can intervene at moment of the simulation. The player can change the Daily Motion pattern of a vessel if he wants to. But after changing the DMP, the system must also generate the history of the “new” DMP. This can be done using one of the two ways described above.

7.4 Database

The sea traffic generator contains a lot of databases to store information about vessels. For our project we need at least the following databases:

- Scenario Databases, which stores all the scenarios.
- Daily Motion Pattern Databases, which stores all Daily Motion Patterns.
- Entity information, which stores information about vessels, like crew, owner, AIS, position and appearance, history.
- Report Database, which stores report about vessels, like incidents report.

There are two ways to fill the database. The first way is to store information which is generated at the start if the simulation, for example the generated history of a vessel. The other way is to fill the databases with information gained from sensors described in section 4.1.4 till 4.1.7. Each sensor observes different type of information within their reachable area. The observed information can be store in a database. If surveillance wants to know more about a vessel he can request information from a certain database.
8. Conclusion

After analyzing lots of papers and research we finally understand how the system will look like. Because we will not develop the whole system ourselves, we are dependent on the work delivered by other employees.

To realize the goal: “generate a realistic world view in a particular situation, with regard to sea traffic”, we needed to understand which components play an important role during the simulation.

The most important factors that are essential to generate realistic sea traffic are:
- Filling the environment of the sea traffic.
- The behavior of the vessels (Daily Motion Patterns).
- The history of vessels.

Filling the environment of the sea traffic is a job for the SIMOBS module. This is coupled to VR-Forces. VR-Forces module is responsible for the graphical user interface and generating objects and events during the simulation.

The behaviors of vessels are based on Daily Motion Patterns. These patterns are stored in a database. These patterns can be modified by the role player of scenario creator. According to DMPs assigned to the vessels, MSA can analyze and detect abnormal behavior. But for the analysis, MSA also need some history information about vessels, to check if the vessels are having suspicious behavior.

To know more about a ship, it is necessary to identify it. This can be done by checking the AIS information. Information observed by different type of sensors can contribute to identify ships or provide extra information about a ship. After analyzing the behavior and information of the vessel, MSA can decide whether action should be taken.

The history of the vessels should be generated at the start of the simulation, based on the assigned DMPs or using the backtracking principle described in section 7.3.

Now we understand more about all components coupled to HLA. And have a clear goal of the Sea traffic generator; we can continue with the next step of the developing process, i.e. the implementation phase. Where we will design the architecture of the system and start implementing it.
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>AIS</td>
<td>Automatic Identification System</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ATC</td>
<td>Agent Technology Centre</td>
</tr>
<tr>
<td>C2</td>
<td>Command and Control</td>
</tr>
<tr>
<td>CGF</td>
<td>Computer Generated Forces</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial-off-the-self</td>
</tr>
<tr>
<td>DMP</td>
<td>Daily Motion Pattern</td>
</tr>
<tr>
<td>Eoil</td>
<td>Electro Optic Infra Led</td>
</tr>
<tr>
<td>FOM</td>
<td>Federation Object Model</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HLA</td>
<td>High Level Architecture</td>
</tr>
<tr>
<td>HMI</td>
<td>Human-Machine Interaction</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent Tutoring System</td>
</tr>
<tr>
<td>KML</td>
<td>Keyhole Markup Language</td>
</tr>
<tr>
<td>M&amp;S</td>
<td>Modeling and Simulation</td>
</tr>
<tr>
<td>MSA</td>
<td>Maritime situation awareness</td>
</tr>
<tr>
<td>OOP</td>
<td>Object Orientated Programming</td>
</tr>
<tr>
<td>RAD</td>
<td>Requirement Analysis Document</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Orientated Architecture</td>
</tr>
<tr>
<td>VR</td>
<td>Virtual Reality</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
References

   Visited on Januari 09, 2012.

   Visited on January 09, 2012.

   Visited on January 09, 2012.

   Visited on: January 03, 2012.

   Paper: SmartObjects and Agents for Autonomous High Population Simulated Environments
   published at: Conference, Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) .
   Place: Waltham, MA

[6] Bolderheij F.
   Paper: Mission-Driven Sensor Management
   Type: Report: Proefschrift.
   Place: Delft

[7] Bourdon S. et al
   Place: Canada
   Organisation: Defence R&D Canada.

[8] CAPT George McGuire RAN
   Combined Maritime Forces. 2009.

[9] Computer Generated Forces
   Organisation: VR-Forces
   Published by: VT MÅK, 2011.
   Visited on: January 03, 2012.

[10] DI-Guy
    Year: 2011.
    Visited on: January 09, 2012.
    URL: http://www.diguy.com/diguy/.

    Paper: A Game AI Approach to Autonomous Control of Virtual Characters
    Published: [Conference] // Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).
[12] Google Java API for KML
URL: http://code.google.com/p/javaapiforkml/.

[13] Hansen M.
Basic SOA Using REST: A Hands-On Guide to Implementing Web services and Service Oriented Architecture

Paper: Cognitive Map-based Modeling and Simulation for Tactical Decision Training
Published: [Conference] // Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).
Place: Columbus, Georgia

Paper: Understanding the Impact of Intelligent Tutoring Agents on Real-Time Training Simulations
Published: [Conference] // Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).
Place: Orlando, Florida

Paper: Pluggable Brains: Separating the Intelligent Behavior from the Simulated Player
Published: [Conference] // Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).
Place: The Hague, The Netherlands

Paper: An Instructor-Centered Declarative Approach to Modeling Synthetic Environments
Published: [Conference] // Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).
Place: The Hague, The Netherlands

[18] Map with ships in realtime - AIS - Ship movement and position
By: Marinetraffic.
Visited on: January 03, 2012.


[20] Pautasso C.
Paper: SOA with REST.
Published by: University of Lugano
Date: March 31, 2009.

Paper: Automated Support for Learning in Simulation: Intelligent Tutoring of Shiphandling
Appendix C: Plan of Approach

Plan of approach
Preface

This is the report of our plan of approach for the Entity Information System. This report is a kind of contract which states the problem that need to be solved during the Sea Traffic Generator project. It records the meets between the client (TNO) and the project members (Nathan and LieYen). Moreover, this report describes the planning of our project. First we will start with an orientation phase, where we perform research about the subject. After this phase we go further with implementation phase, where we start designing the system that needs to be implemented, after design the architecture we start implementing it. We have different kind of milestone where the system should meet parts of the requirements, that we have record with the client. For every milestone we will test the code automatically and manually using CPPUNIT, to gain the optimal result and improve it in the next milestone. In the final milestone the system should be done and meets all requirement agreed between the client and us.

Unlike most of the students in our study, we are running this project part-time instead of full-time. This has been agreed between our supervisors and the client. Normally a student requires doing a project within 10 or 11 weeks, but since we are part-timers, we have about 22 weeks to develop the whole system, based on 3 days a week. Starting the internship at January 02, 2012 till June 13, 2012. Within the six months we took out several weeks to have a break, because of school exams.

Since we have some delay in getting the VOG(certificate of good behavior), the first few weeks we started to work at the university, to prevent the project having more delay. So the after our exam week we hope we can start running the project at the TNO.
# Table of Contents

1. Introduction ......................................................................................................................... 96

2. Project description ................................................................................................................ 97
   2.1 Project environment ........................................................................................................... 97
   2.2 Purpose .............................................................................................................................. 97
   2.3 Assignment ....................................................................................................................... 97
   2.4 Deliverable products ........................................................................................................... 97
   2.5 Requirements and Limitations .......................................................................................... 98
   2.6 Preconditions ..................................................................................................................... 98

3. Approach and planning ......................................................................................................... 99

4. Project design ....................................................................................................................... 101
   4.1 Organization ...................................................................................................................... 101
   4.2 Administrative procedures ................................................................................................. 101
   4.3 Funding .............................................................................................................................. 101
   4.4 Reporting ........................................................................................................................... 101

5. Quality assurance ................................................................................................................ 103
1. Introduction

The Sea Traffic Generator project arises to facilitate the research developing tools for automatic maritime behavior analysis. To develop a realistic simulator, it is useful to work with a virtual world in which a scenario can be build close to reality. But it is very labor-intensive to build a large simulation containing a realistic environment. Even using a scenario generator, because all scenarios are inserted in the system manually and the environment is still not filled in realistic enough. So a system that is able to generate sea traffic automatically based on Daily Motion Patterns is desired.

This report should be approved by our supervisor Robbert Krijnen. After changes and agreeing this report we will continue the activities that are described in this report.

In this report we explain our approach of the Entity Information System. First we describe details about the project itself, like the purpose, the client, deliverable products and preconditions. Chapter three describes our approach to the project. After that some details will be provided about the project design in Chapter four. In the last chapter we discuss the quality assurance.
2. **Project description**

This chapter provides more information about the project itself. We explain the desired changes of the system, so it can meet the requirements of the client. Another purpose of this chapter is to get a clear overview what the actual requirements of the system are, that needs to be satisfied.

2.1 **Project environment**

The idea to develop the Entity Information System results from a previously developed system called SIMOBS. This system is an application which you can use to simulate an environment with villagers having specific behavior.\(^\text{14}\) But now TNO would like to expand this system, so it can also be used for simulating a realistic representation of the sea traffic.

2.2 **Purpose**

The purpose of the system is to create a consistent worldview with associated history, based on Daily Motion Patterns. Subsequently adapt the software, so new techniques can be added, leading to new insights. Besides the above mentioned purposes, the system could also be used for training people who observe sea traffic and suspicious behavior on sea.

2.3 **Assignment**

The assignment can be formulated as follow: “Create a history generator for an entity (vessel), so the simulation can be as realistic as the real world”. So we must do some research about to generate the history of these entities, if AIS, DMP, sensor information is given. The history of each entity should be as realistic as possible.\(^\text{15}\) According to the observation of sea traffic, analyzers should be able to detect suspicious behavior. Our main assignment is to consider how history of entities can be generated as realistic as possible, based on the assigned Daily Motion Pattern.

2.4 **Deliverable products**

At the end of the project a working prototype of the Entity Information System must be completed. The system should be able to generate history of a vessel, based on its Daily Motion Pattern and AIS information\(^\text{16}\). It should be desirable if the system also allows a role player to intervene in the simulation, where the player can modify a vessel or scenario. A High-Level-Architecture(HLA) will be used as communication medium between de pluggable parts like VR-Forces, AIS Real-Time Generator.

---

\(^\text{14}\) For more information about SIMOBS we refer to our Orientation Report section 4.2.3
\(^\text{15}\) The (original) detailed project description can be found in our Appendix A of our Orientation Report.
\(^\text{16}\) More information about Daily Motion Pattern, AIS, and HLA can be found in our Orientation Report chapter 4
2.5 Requirements and Limitations

The Entity Information System should meet lots of requirements. The requirements can be found in our Requirement Analysis Document (RAD). This RAD describes the functional and nonfunctional requirements that the system can meet. According to the MoSCoW model\textsuperscript{17}, the requirements will be sorted on priority. The requirements can be divided into four groups. The first group contains requirements that the system must meet. Followed by requirements that the system should meet. The third and fourth group contains requirements that are desirable if the system can meet, but these requirements have the least priority.

Moreover, the system should be implemented on a way that is can also be used for other purposes beside of Sea Traffic. For example the system should also be possible to use for air traffic. With other words the system should be able to be reused for other projects.

2.6 Preconditions

This section describes the most important preconditions of the system to achieve the main goal of the project. The preconditions are mentioned below:

- The Daily Motion Patterns should be described as realistic as possible, in order to generate history that is as realistic as possible. Because the history of entities are generated are based on the assigned Daily Motion Patterns.

- To cover the requirements, we will first develop a working basic prototype containing the basic functionality. Based on this basic prototype we will continue developing the system and expand the functionality, so it will cover as many requirements as possible at the end of our project.

\textsuperscript{17} Our MoSCoW Model can be found in chapter 3 of our Requirement Analysis Document.
3. Approach and planning

The developing process can be divided into three phases. The first one is the orientation phases, which means we do research about the problem and analyze papers to obtain more information, which can help us developing the system. The second phase is the design phase. In this phase we will design the architecture of the system and based on this architecture we will start implementing the system. The result of this phase will be reported in the architectural design document. The last phase is the implementation and test phase, in this phase we will test the implemented system. Actually we will constantly repeat the second and third phase, because after testing each milestone we will try to modify the system, so it meets the requirements for the next milestone as well as possible. Bases on this iterative process we will try to improve the system with every milestone.

Since we are part timely developing this system, we have about 22 weeks before the entire system should be done. We will spend the first few weeks for doing some literature study and research. After we have enough knowledge about the system we would like to develop, we will start designing the system in details, which result in the design phase. In week eight we will start designing the system, based on the created architecture in the week before. After four weeks we should have implemented a basic working prototype. And then the whole process of design and implementation starts over again, to update and improve the system.

A short overview of our planning created on 18 January 2012:

<table>
<thead>
<tr>
<th>Week 1 till week 3:</th>
<th>Literature study</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Analyzing papers</td>
</tr>
<tr>
<td></td>
<td>- Write orientation report (OR)</td>
</tr>
<tr>
<td></td>
<td>- Write requirement analysis document (RAD)</td>
</tr>
<tr>
<td>Week 4 and week 5:</td>
<td>Break, because of school exams.</td>
</tr>
<tr>
<td>Week 6 till week 8:</td>
<td>From this week working 3 days a week at TNO.</td>
</tr>
<tr>
<td></td>
<td>- Processing feedback on RAD and OR.</td>
</tr>
<tr>
<td></td>
<td>- Writing documentation and designing system.</td>
</tr>
<tr>
<td></td>
<td>- Describe use cases and scenarios.</td>
</tr>
<tr>
<td></td>
<td>- MoSCoW document</td>
</tr>
<tr>
<td>Week 8 till week 13:</td>
<td>- Architectural Design Document</td>
</tr>
<tr>
<td></td>
<td>- Technical Design Document</td>
</tr>
<tr>
<td></td>
<td>- Implementation plan</td>
</tr>
<tr>
<td></td>
<td>- Develop system</td>
</tr>
<tr>
<td>End of Week 13</td>
<td>- First milestone: a basic working prototype</td>
</tr>
<tr>
<td></td>
<td>- Update the related documents according to the milestone.</td>
</tr>
<tr>
<td>Week 14</td>
<td>This week working 2 days instead of 3, because of preparation school exams.</td>
</tr>
<tr>
<td></td>
<td>- Update the related documents according to the milestone.</td>
</tr>
<tr>
<td></td>
<td>- Improve system by adding new functionalities</td>
</tr>
<tr>
<td>Week 15</td>
<td>Break, because of school exams.</td>
</tr>
<tr>
<td>Week 16</td>
<td>This week working 2 days instead of 3, because of preparation school exams.</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>- Update the related documents according to the milestone.</td>
</tr>
<tr>
<td></td>
<td>- Improve system by adding new functionalities</td>
</tr>
<tr>
<td>Week 17 till week 21</td>
<td>- Improve system</td>
</tr>
<tr>
<td></td>
<td>- Modify documentation according to each milestone.</td>
</tr>
</tbody>
</table>

From week 8 on, we will also be working on the final report. To keep the necessary documentation updated each time we improved the system. The last 2 weeks we will also spent some time to prepare for the presentation.

A detailed scheme of the planning can be found in Appendix A: planning.
4. **Project design**

This chapter provides the visualization of how the project manager intends to set up the project according to the proposed approach.

4.1 **Organization**

In this part we will describe the supervisors can coaches who help us develop to develop the system and guide us during the project.

- Mr. M.Sc. R. Krijnen; Modeling, Simulation & Gaming; TNO Defense and Safety Supervisor TNO;
- Mr. M.Sc. P. Kerbusch; Modeling, Simulation & Gaming; TNO Defense and Safety Artificial Intelligence expert TNO;
- Mr. Dr. Ir. R. Smelik ; Modeling, Simulation & Gaming; TNO Defense and Safety VR Forces/SIMOBS expert TNO;
- Mr. Ir. R.J.E. Jansen; Modeling, Simulation & Gaming; TNO Defense and Safety HLA expert TNO;
- Mr. E. Breijers; TNO Defense and Safety Domain expert TNO;
- Mr. Dr. ir. G. de Haan; Visualization and Virtual Reality; TU Delft Supervisor/mentor Delft University of Technology;

4.2 **Administrative procedures**

Before we can start working at our offices in TNO, we should request VOG (certificate of properly behavior) first. Without VOG we are not allowed to walk into the building without a supervisor of TNO. Moreover, we also had to sign a contract, which states the terms and conditions between TNO and Delft University of Technology.

4.3 **Funding**

Funding is not applicable in our project, because all the software and tools we need will be provided by TNO. Even the workspace and the computer are regulated by TNO.

4.4 **Reporting**

We will have a meeting with our supervisor of Delft University of Technology, once in 3 or 4 weeks. To report what we have done and how the project is running. The results of these meetings can be found in the *Two weekly process reports*. 
On the other side we will be working at TNO Den Haag, so the supervisor there can come to our offices at any time to check the process. Moreover, each phase of the development will result in a report, which we will handle to both of the supervisors.

The project ends with a report, presentation about the system. Last but not least a working system that meets the established requirements should be developed.
5. Quality assurance

The quality of the process is ensured by systematically working with a planning/schedule. Regular progress meetings with the supervisor and employees shall ensure that these plans are effectively monitored and that any bottlenecks removed. These discussions will take place at least once a week. During the execution of the project we will keep logs in which each day’s work that day will be recorded.

The quality of the system is ensured using software development methods recommended by Technology University of Delft and literature study. This means dividing the software development process into three phases described in Chapter 3. This leads to a better development of the system, because we first have to do some analysis and research about the subject, following by designing the system and finally testing it. This process is step by step gaining more information to improve the system.

Additionally, we control quality assurance by showing our supervisor what we have been done weekly, when we are having a meeting. All feedback will be carefully evaluated and implemented or countered by a contradictory argument.

Testing will be done by unit testing all methods through assertions, using Visual Studio’s standard testing environment. If we have any questions or bottlenecks, we can also knock on the door of some TNO experts, who can explain and help us.
### Appendix A: Planning

The planning created on 18 January, 2012
Appendix D: Requirement Analysis Document

Requirement Analysis Document
# Table of Contents

1. Introduction .................................................................................................................. 107

2. Requirements ................................................................................................................. 109
   2.1 Functional Requirements ......................................................................................... 109
       2.1.1 General............................................................................................................ 109
       2.1.2 Sea Traffic Generator (STG) ............................................................................ 113
       2.1.3 Maritime Situation Awareness (MSA).............................................................. 115
       2.1.4 Diagram.......................................................................................................... 119
   2.2 Non-Functional Requirements .................................................................................. 121
       2.2.1 Usability .......................................................................................................... 121
       2.2.2 Reliability ........................................................................................................ 121

3. Scenarios & Use Cases ................................................................................................. 122
   3.1 Actors ....................................................................................................................... 122
   3.2 Scenarios .................................................................................................................. 123
       3.2.1 USER – SYSTEM ............................................................................................ 123
       3.2.2 SYSTEM – SYSTEM ...................................................................................... 123
   3.3 Use cases ................................................................................................................. 124
       3.3.1 USER - SYSTEM ............................................................................................ 124
       3.3.2 System-System ............................................................................................... 132
1. Introduction

Currently the creation of a realistic scenario to achieve effective training and education or analysis in a Virtual Environment takes too much time. Until now, the focus lies on the particular behavior, while less attention is paid to fill this worldview, by entities with normal behavior. The lack of it leads to our problem statement: “generating a realistic view of the world takes too much time”. By making use of a division of abstraction levels, we will develop a solution of the problem statement using related requirements; generating a realistic scenario in general, generating a realistic focused on the maritime, called the Sea Traffic Generator (STG) and generating a realistic scenario based on the requirements for the Maritime Situation Awareness (MSA) research project.

In general, we will focus on the basic aspects of this problem. The foundation of the scenario-creation and entity behavior based on certain patterns has to be included. With this general part, it is possible to use the generator for multiple purposes, which implies reusability. The requirements based on this part are listed in section 2.1.1.

An application of this kind of generator is the Sea Traffic Generator, focused on the maritime. The idea to develop the Sea Traffic Generator results from the previously developed system called Simobs\textsuperscript{18}. With this system it is possible to create a simulation, with generated people, based on a Pattern of Life also called Daily Motion Patterns (DMPs). For example, what time some villagers are going to their work or going back to their home is among others defined in this pattern.

To develop a realistic simulator, it is useful to work with a virtual world in which a scenario can be build close to reality. The scenario must contain the full range of ships, ports, fishing areas, routes etc. To generate a realistic output, so called Daily Motion Patterns (DMPs), will be used. These patterns have a similar behavior as the Pattern of Life. It simulates a realistic behavior per type of ship, with a build-in variation per ship to transform every DMP to a unique pattern per vessel. DMPs can also be used to generate occasional suspicious behavior. DMPs are not only used to describe behavior, but they can also be used to generate history.

One of the main goals of the Sea Traffic Generator is to provide a realistic appearance of the real world. It simulates a real world, based on realistic DMPs. For validation of analysis algorithms, researchers wants to have a comparison between this simulated real world and the perceived world, the world they can actually perceive by use of sensors. This perceived world is a subset and possible distorted view of the real world, because (of course) not everything can be observed by sensors and not everything is observed correctly. For example: a sensor observes a ship on time \( X \) at position \( Y \), we want to know if we can gather more information about the ship to do behavior analysis. The “more” information could come from other sources or from history information. Another example; a ship always travels at a certain route. However once a month he shows different behavior! Why is that? The requirements related to the STG are elaborated in section 2.1.2.

In order to analyze Maritime Situation Awareness (MSA), the research and development department (R&D) for the Maritime Behavior Analysis has large databases with real vessel traffic information. But this data is on its self not sufficient and has no reference with the ground truth, by

\textsuperscript{18} For more information about SIMOBS please read the Orientation Report Ch. 4.2.4
multiple reasons; AIS\textsuperscript{19} data only contains large vessels, experiments with small vessels also provide an incomplete image of the reality. Using this real data it is also hard to influence a specific situation or scenario for analysis purposes. For example: a situation occurring in the Noordzeee cannot be generalized to a similar situation in the Gulf of Aden, where takes place. By using this data and the expertise of the Coast Guard, it is possible to develop relevant DMPs to obtain a realistic generation and simulation of the reality. By keeping track of the history of the vessels, one can perform several investigations. Requirements associated with MSA can be found in section 2.1.3.

Chapter 3 will describe the extended use cases based on some requirements of section 2.1.

\textsuperscript{19} For detailed information we refer to the Orientation Report Ch. 4.1.4
2. Requirements

In this part we describe the different types of requirements, categorized by functional and non-functional requirements.

2.1 Functional Requirements

The requirements can be divided in three main (abstraction) parts; General, Sea Traffic Generator (STG) and Maritime Situation Awareness. Each part contains an abstraction level. In the general part, we describe the basic requirements for scenario building. Second, in the STG-part, we will take a closer look at the specific real time simulation of sea traffic. Finally, the MSA conditions are established as requirements, specific for the purposes of research and analysis. We will use a coverage diagram (figure 1) to prevent repetition of requirements in the several abstraction parts. If a requirement repeats in order to specify more, we will make use of an extension and only describe the specification. Using a diagram in section 2.1.4 we will clarify the inheritance and overlap between the parts on a more detailed level.

2.1.1 General

Basic scenarios, with less entities will not provide a sufficient projection of the reality. In general the need of fast and simple scenario creation must be provided. In order to do this, we have to set up requirements as foundation. The two main part of this section are the geographic set and the entities.

General requirements

<table>
<thead>
<tr>
<th>REQ</th>
<th>IDENTIFICATION</th>
<th>DESCRIPTION</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ: The system must be able to create area’s and assign one or multiple functions to it</td>
<td>REQ_GEN_001</td>
<td>Specify an area and assign one or multiple functions to it in order to give this area specific properties. For example: urban area, industrial area, etc.</td>
<td></td>
</tr>
<tr>
<td>REQ: The system must be able to create routes and assign one or multiple functions to it</td>
<td>REQ_GEN_003</td>
<td>Specify a route and assign one or multiple functions to it in order to give this route specific properties. For example: road, sea lane, air lane etc.</td>
<td></td>
</tr>
<tr>
<td>REQ: The system must be able to generate entities for a route</td>
<td>REQ_GEN_004</td>
<td>The system should be able to generate different kind of entities for a route based on its properties (a.k.a. template based entity generation). Entities generated for the specific route will use Daily Motion Patterns (DMPs) to simulate their behavior. The generated entities are not limited to their initial route. This generation process must be defined in template-files. For example: to fill a shipping lane with two cargo vessels and one cruise ship.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1. Requirement Coverage
**REQ:** The system must be able to create spawn points and assign one or multiple functions to it  
**IDENTIFICATION:** REQ_GEN_005  
**PURPOSE:** Create a point from where entities can be generated.  
**DESCRIPTION:** Specify a point and assign one or multiple functions to it, in order to give this point specific properties.  
For example: factory, etc.

**REQ:** The system must be able to generate entities for a spawn point  
**IDENTIFICATION:** REQ_GEN_006  
**DESCRIPTION:** The system should be able to generate different kind of entities from a point based on its properties (a.k.a. template based entity generation). Entities generated from the specific route will use Daily Motion Patterns (DMPs) to simulate their behavior for the specified route. The generated entities are not limited to their initial point. This generation process must be defined in template-files.

**REQ:** The system must be able to create points from where entities will be removed  
**IDENTIFICATION:** REQ_GEN_007  
**PURPOSE:** Create a point from where entities will be removed.  
**DESCRIPTION:** Specify a point and assign a removing function to it in order to give this point specific properties. There must be a connection (route) between the spawn points (REQ_GEN_005) and the removal points.  
For example: remove traffic from a route that’s outside of the area of interest for the scenario.

**REQ:** The system must have the possibility to assign density to an area or route  
**IDENTIFICATION:** REQ_GEN_008  
**DESCRIPTION:** The area, route, described in REQ_GEN_001 until REQ_GEN_004, must be able to have a certain density level. This density level is the amount of certain entities in this area or route.  
For example: number of houses, population, traffic. The amount of habitant per square meter is much higher in Holland than in Australia.

**REQ:** The system must store entity information in a data store.  
**IDENTIFICATION:** REQ_GEN_009  
**PURPOSE:** To save information about an entity (or multiple entities).  
**DESCRIPTION:** It should be possible to save information about one or multiple entities.  
For example: ‘Store a ships its position and time in the database twice a day for analysis purposes’

**REQ:** The user must be able to request information from a data store using queries.  
**IDENTIFICATION:** REQ_GEN_010  
**PURPOSE:** To gain information about an entity (or multiple entities).  
**DESCRIPTION:** By using a data-requesting component, it is possible to obtain information about one or multiple entities. It should be possible to request these information with a query by using conditions.  
For example: ‘I want to know the route of a vessel in the past three days’

**REQ:** The system must be able to initialize a scenario based on a random start time  
**IDENTIFICATION:** REQ_GEN_011  
**PURPOSE:** Provide an initialization of the scenario. Like a fishing area is already filled with fishing boats dependent of the start time. Or a road has a traffic density based on the start time  
**DESCRIPTION:** Run the simulation with a random initial time. All entities should have a start position based on the initialization time.
REQ: The system must have the possibility to import geographic areas, roads and points.
IDENTIFICATION: REQ_GEN_012
PURPOSE: To create a scenario, quickly.
DESCRIPTION: The system is able to import geographic areas, roads and points.
For example: geographical information of shipping lanes, streets, urban area’s.

REQ: The system must have the possibility to save the created scenario
IDENTIFICATION: REQ_GEN_013
PURPOSE: To reuse a previously created scenario, quickly, so that we do not have to sketch the scenario again.
DESCRIPTION: The system is able to save the created scenario, with options like the geographic areas, the selected areas, the selected routes, assigned properties and the inserted entities.

REQ: The scenario must contain an area of interest
IDENTIFICATION: REQ_GEN_014
DESCRIPTION: An area of interest can be indicated. The area of interest is a region in which the scenario takes place. Outside this area entities and behavior does not have to be simulated.
For example: the whole area between the coast of Holland and the sea border between the Holland and Great Britain.

REQ: The system must be able to accelerate the simulation speed.
IDENTIFICATION: REQ_GEN_015
PURPOSE: To avoid a long wait, before an event occurs.
DESCRIPTION: The simulation speed can be controlled manually.

Daily Motion Pattern requirements

REQ: The system must simulate entity behavior, based on DMPs
IDENTIFICATION: REQ_GDP_001
DESCRIPTION: The system must simulate the entities, based on Daily Motion Patterns.
For example: simulation of a ferry, containing its positions and destinations in a way it matches with a realistic route of a ferry.

REQ: DMPs must be described using finite state machine
IDENTIFICATION: REQ_GDP_002
DESCRIPTION: The DMPs must support Finite State Machine to describe its behavior.

Entity requirements

REQ: The system must have an Entity generator
IDENTIFICATION: REQ_GNT_001
DESCRIPTION: The entity generator (templates) must support the following aspects:
- Density: minimum and maximum number of entities (with a certain type) to be generated.
- Distribution: distribution of entity types to be generated.
- Entity: a variance in generation time (generate every 10 minutes with a 1 minute variance)
- Entity: support the concept of be at location X at a time Y, using DMP Z.
For example: During a simulation we want to have 10 entities with type 1 in the same area performing their job described in their DMP.

REQ: The entity generator must have a varying output
**IDENTIFICATION**: REQ_GNT_002

**DESCRIPTION**: The generation of entities is either with or without random seed. *For example: When a simulation is running 10 different times, each time the initialization of the entities can be exact the same or the initialization can vary from the previous time.*

**REQ**: The DMPs must have a persistent behavior

**IDENTIFICATION**: REQ_GNT_003

**DESCRIPTION**: The DMP is either with or without random seed. *For example: When a simulation is running 10 different times, each time the DMPs in the simulation can be exact the same or the DMPs can vary from the previous time.*

### Scenario builder requirements

**REQ**: The user must have the possibility to **execute** REQ_GEN_001

**IDENTIFICATION**: REQ_GSB_001

**DESCRIPTION**: The scenario builder is able to create area’s and assign one or multiple functions to it.

**REQ**: The user must have the possibility to **execute** REQ_GEN_003

**IDENTIFICATION**: REQ_GSB_002

**DESCRIPTION**: The scenario builder is able to create routes and assign one or multiple functions to it.

**REQ**: The user must have the possibility to **execute** REQ_GEN_005

**IDENTIFICATION**: REQ_GSB_003

**DESCRIPTION**: The scenario builder is able to create spawn points and assign one or multiple functions to it.

**REQ**: The user must have the possibility to **execute** REQ_GEN_007

**IDENTIFICATION**: REQ_GSB_004

**DESCRIPTION**: The scenario builder is able to create (sync)points from where entities will be removed.

**REQ**: The user must have the possibility to **execute** REQ_GEN_012

**IDENTIFICATION**: REQ_GSB_005

**DESCRIPTION**: The scenario builder is able to import geographic areas, roads and points.

**REQ**: The user must have the possibility to **execute** REQ_GEN_013

**IDENTIFICATION**: REQ_GSB_006

**DESCRIPTION**: The scenario builder is able to save the created scenario.

**REQ**: The user must have the possibility to **execute** REQ_GEN_015

**IDENTIFICATION**: REQ_GSB_007

**DESCRIPTION**: The scenario builder is able to accelerate the simulation speed.

### Role Player requirements

**REQ**: The user should be able to add entities manually.

**IDENTIFICATION**: REQ_GRP_001

**DESCRIPTION**: Possibility to add entities and assign a DMP manually. *For example: adding a cargo vessel to the harbor of Rotterdam.*
**REQ:** The user should be able to change DMP of an entity.

**IDENTIFICATION:** REQ_GRP_002

**DESCRIPTION:** The DMP assigned to an entity should be able to change.

*For example: The user wants a fishing ship, to change its behavior to one of a drug smuggler.*

**REQ:** The user must be able to disconnect DMPs of entities

**IDENTIFICATION:** REQ_GRP_003

**DESCRIPTION:** The user must be able to disconnect entities behavior, based on DMPs, to manual control the entity in VR-Forces GUI.

*For example: A user wants to manually influence the simulation, which was based on the DMPs.*

**REQ:** The user must be able to manage the simulation speed (REQ_GEN_015)

**IDENTIFICATION:** REQ_GRP_004

**DESCRIPTION:** The user must be able to manage the simulation speed in order to increase the simulation at certain moments.

## 2.1.2 Sea Traffic Generator (STG)

This section describes the requirements for the Sea Traffic Generator. It should be able to generate vessels and simulates their behavior, implemented in Daily Motion Patterns. This pattern ensures that the vessels has a realistic sailing behavior, like for instance a fishing boat sails from a harbor to a fishing area or a merchant ship travels to and from a harbor using sea lanes.

Mainly, the goal of the STG is to generate the complete *real world* besides the *perceived world*. By using observing techniques, like sensors, a perceived world can be made. Based on this information you get a incomplete scenario. The STG will be used to generate the missing world, which is not observed besides the perceived world.

### General requirements

**REQ:** The system must have the possibility to import geographic areas, roads and points

**EXTENSION OF:** REQ_GEN_012

**IDENTIFICATION:** REQ_STG_001

**SPECIFICATION:** At most 100 geographic context areas.

**REQ:** The scenario must contain an area of interest

**EXTENSION OF:** REQ_GEN_014

**IDENTIFICATION:** REQ_STG_002

**SPECIFICATION:** An area size of at least 500 by 1000 km, should be handled by the system.

**REQ:** The system must be able to accelerate the simulation speed

**EXTENSION OF:** REQ_GEN_015

**IDENTIFICATION:** REQ_STG_003

**SPECIFICATION:** The simulations speed should be possible to vary the speed between 1 and 60 times.

**REQ:** A vessel could have a owner

**IDENTIFICATION:** REQ_STG_004

**DESCRIPTION:** A vessel can have an owner, which is defined.
<table>
<thead>
<tr>
<th><strong>REQ:</strong></th>
<th>A owner could own multiple ships</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IDENTIFICATION:</strong></td>
<td>REQ_STG_005</td>
</tr>
<tr>
<td><strong>PURPOSE:</strong></td>
<td>To analyze the interchange of crew, the owner possesses</td>
</tr>
<tr>
<td><strong>DESCRIPTION:</strong></td>
<td>A owner of a ship, could also own other ships.</td>
</tr>
</tbody>
</table>

**Daily Motion Pattern requirements**

<table>
<thead>
<tr>
<th><strong>REQ:</strong></th>
<th>The system must simulate vessel behavior, based on several DMPs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXTENSION OF:</strong></td>
<td>REQ_GDP_001</td>
</tr>
<tr>
<td><strong>IDENTIFICATION:</strong></td>
<td>REQ_SDP_001</td>
</tr>
<tr>
<td><strong>SPECIFICATION:</strong></td>
<td>The system must simulate the following vessel types using Daily Motion Patterns.</td>
</tr>
<tr>
<td></td>
<td>- Ferry</td>
</tr>
<tr>
<td></td>
<td>- Fishing boat</td>
</tr>
<tr>
<td></td>
<td>- Trawler</td>
</tr>
<tr>
<td></td>
<td>- Merchant ship</td>
</tr>
<tr>
<td></td>
<td>- Pleasure cruise</td>
</tr>
</tbody>
</table>

**Vessel requirements**

<table>
<thead>
<tr>
<th><strong>REQ:</strong></th>
<th>The system should support different types of vessels.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IDENTIFICATION:</strong></td>
<td>REQ_SVE_001</td>
</tr>
<tr>
<td><strong>PURPOSE:</strong></td>
<td>Different types of vessels, to vary the scenario.</td>
</tr>
<tr>
<td><strong>DESCRIPTION:</strong></td>
<td>The system should be able to generate different types of vessels.</td>
</tr>
</tbody>
</table>

**Information requirements**

<table>
<thead>
<tr>
<th><strong>REQ:</strong></th>
<th>The system should be able to store information of the ships.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXTENSION OF:</strong></td>
<td>REQ_GEN_009</td>
</tr>
<tr>
<td><strong>IDENTIFICATION:</strong></td>
<td>REQ_SHI_001</td>
</tr>
<tr>
<td><strong>PURPOSE:</strong></td>
<td>The possibility to use it for analysis.</td>
</tr>
<tr>
<td><strong>SPECIFICATION:</strong></td>
<td>It should be possible to store information about the vessel in an data storage system.</td>
</tr>
<tr>
<td></td>
<td>For example you can think of INTELL report, AIS, sensor information (RCS and EO/IR), picture information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>REQ:</strong></th>
<th>The user must be able to request data from an information-system using queries.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXTENSION OF:</strong></td>
<td>REQ_GEN_010</td>
</tr>
<tr>
<td><strong>IDENTIFICATION:</strong></td>
<td>REQ_SHI_002</td>
</tr>
<tr>
<td><strong>PURPOSE:</strong></td>
<td>To gain history about one or multiple vessels.</td>
</tr>
<tr>
<td><strong>SPECIFICATION:</strong></td>
<td>By using a module, it is possible to trace history about one or multiple vessels.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>REQ:</strong></th>
<th>The system should be able to generate history, based on DMPs.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IDENTIFICATION:</strong></td>
<td>REQ_SHI_003</td>
</tr>
<tr>
<td><strong>DESCRIPTION:</strong></td>
<td>It should be possible to generate history about the vessel in an information-system.</td>
</tr>
<tr>
<td></td>
<td>For example you can think of INTELL report, AIS, sensor information (RCS and EO/IR), picture information.</td>
</tr>
</tbody>
</table>
2.1.3. Maritime Situation Awareness (MSA)

More specific, the Maritime Situation Awareness is more specialized to analyze behavior. This is made possible by logging the complete history of the ships and the possibility to influence the simulation by the injection of suspicious behavior. The requirements regarding to the logging are described in History, the injection of behavior and manipulation of the simulation is described in Role Player.

In the MSA research developments sensor fusion & tracking, information fusion, behavioral analysis and C2 sensor (platform) management profit of the increased (detection) capacity and (surveillance) coverage of the sensor systems. In Maritime surveillance sensor on land, at sea, underwater, in the air and space borne are used. Platforms carrying sensors can be manned or unmanned (e.g. UAS/UAV).

**Daily Motion Pattern requirements**

<table>
<thead>
<tr>
<th>REQ:</th>
<th>The system must simulate vessel behavior, based on several DMPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension of:</td>
<td>REQ_SDP_001</td>
</tr>
<tr>
<td>Identification:</td>
<td>REQ_MDP_001</td>
</tr>
<tr>
<td>Specification:</td>
<td>Based on the DMP, the system simulates a realistic behavior of the vessels. Also suspicious behavior is implemented in DMPs.</td>
</tr>
</tbody>
</table>

| REQ: | After a DMP modification, the system stores history based on the new DMP (using REQ_GRP_002). |
| Identification: | REQ_MDP_002 |
| Description: | When modifying an entity, the new history stored from the change will be based on the new DMP. Then the entity must behave according to this DMP. |

| REQ: | The system could reuse DMPs when operating in a different area |
| Identification: | REQ_MDP_003 |
| Purpose: | To increase the reusability of the software. |
| Description: | When operating in another area, some entities will also occur here. It is possible to reuse the DMPs, for the same entities, like fishing boats, by only modify the start and destination. A lot of behavior will be the same as the fishing boats in another area. |

| REQ: | Every vessel should have a unique behavior |
| Identification: | REQ_MDP_004 |
| Purpose: | To create a unique behavior per entity. |
| Description: | This is achieved by using variables, the degrees of freedom, and the DMPs of the vessel type. These variables will provide a variation within the DMPs, what gives a unique behavior per ship. Variables are: day/night (time of the day), home harbour, typical destination(s), typical speed, typical cargo, variation is speed (Rush/rush vs relaxed), variation in trajectory, variation in time, variation in cargo. |

**Role Player requirements**

| REQ: | The actions of the role player should be stored. |
| Identification: | REQ_MRP_001 |
| Description: | If a role player modifies the simulation, a record should be stored about the actions of the role player. |
**REQ:** The role player should be able to publish an INTELL report.
**IDENTIFICATION:** REQ_MRP_002
**DESCRIPTION:** The role player executes an INTELL report to create suspicious behavior. When the role-player user selects a ship, and INTELL report can be linked with this ship.

### History requirements

**REQ:** The system should be able to provide a complete log for the Sea Traffic Generator
**EXTENSION OF:** REQ_HSI_002
**IDENTIFICATION:** REQ_MHI_001
**PURPOSE:** To compare this log in the MSA with the **MSA Performance Module**
**DESCRIPTION:** The log contains information about:
- All (currently present) entities (position, course, speed, cargo, ...) **REQ_AIS_001**
- Set of INTELL-reports **REQ_INT_001**
- The whole history (1-3 month) **REQ_MVE_004**
- The used DMPs per entity **REQ_MVE_001**

- **Sensor Information Generator**
  **REQ:** The user must be able to request Radar Cross Section information about an entity.
  **EXTENSION OF:** REQ_SHI_002
  **IDENTIFICATION:** REQ_RCS_001
  **PURPOSE:** Provide signature information based on their *appearance, time and location*, to determine entity-type.

  **REQ:** The user must be able to request EO / IR information about an entity from a table.
  **EXTENSION OF:** REQ_SHI_002
  **IDENTIFICATION:** REQ_EOI_001
  **PURPOSE:** Provide signature information based on their *appearance, time and location*, to determine entity-type.

- **AIS Information Generator**
  **REQ:** The user must be able to request AIS information for one or multiple entities.
  **EXTENSION OF:** REQ_SHI_002
  **IDENTIFICATION:** REQ_AIS_001
  **PURPOSE:** To identify an entity, based on their *position, speed, mission, port of destination*.

  **REQ:** The system should be able to generate specific AIS information about an entity.
  **EXTENSION OF:** REQ_SHI_003
  **IDENTIFICATION:** REQ_AIS_002
  **PURPOSE:** To build a history of the entity.
  **DESCRIPTION:** The system should be able to generate the past of a vessel based on AIS information. Specific parameters are: *position, speed, mission, port of destination*.

- **Picture Information Generator**
  **REQ:** The user must be able to request wiki information about an entity
  **EXTENSION OF:** REQ_SHI_002
  **IDENTIFICATION:** REQ_PIC_001
  **PURPOSE:** The user must be able to request provide a name and/or MMSI of a specific entity based on their *picture*. 
INTELL Information Generator

**REQ:** The user must be able to request INTELL report information by entity’s name or MMSI

**EXTENSION OF:** REQ_SHI_002

**IDENTIFICATION:** REQ_INT_001

**PURPOSE:**
1. To check whether the entity is suspicious
2. Persistent surveillance

Alpha Report

**REQ:** The user should be able to request one or multiple Alpha report(s)

**EXTENSION OF:** REQ_SHI_002

**IDENTIFICATION:** REQ_MAR_001

**PURPOSE:** To mention that a ship is located in a secured shipping lane

**DESCRIPTION:** By selecting vessel(s), related Alpha report(s) can be requested. This report contains: the cargo type, the (passage) speed and freeboard (the distance between waterline and first open deck).  

**REQ:** The system should be able to store Alpha report(s)

**EXTENSION OF:** REQ_SHI_001

**IDENTIFICATION:** REQ_MAR_002

**DESCRIPTION:** The generated Alpha report(s) (REQ_MAR_003), should be stored in a independent data store.

**REQ:** The system should be able to generate Alpha report(s)

**EXTENSION OF:** REQ_SHI_003

**IDENTIFICATION:** REQ_MAR_003

**DESCRIPTION:** When a vessel is using a secured area, it sends its position with a certain interval in an Alpha report. This report should be generated by the system.

Vessel

**REQ:** The user must be able to request the log of the entity-type modifications

**EXTENSION OF:** REQ_SHI_002

**IDENTIFICATION:** REQ_HMV_001

**DESCRIPTION:** The user must be able to request a log, based on a specific ship, with the modifications of vessel type during the simulation, based on DMPs, from the database.

**REQ:** The system must be able to keep a log of the entity-type modifications

**EXTENSION OF:** REQ_SHI_001

**IDENTIFICATION:** REQ_HMV_002

**DESCRIPTION:** The system must be able to keep a log of changes, when the user changes the entity type of a vessel, in the database.

**REQ:** The user must be able to request stored position(s) of the ships

**EXTENSION OF:** REQ_SHI_002

**IDENTIFICATION:** REQ_HMV_003

**DESCRIPTION:** The user must be able to request the position(s) of one or multiple vessels, the vessel sailed during the simulation, or based on the generated history.

**REQ:** The system must be able to store position(s)

**EXTENSION OF:** REQ_SHI_001
<table>
<thead>
<tr>
<th>IDENTIFICATION: REQ_HMV_004</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION: The system must be able to store the position of the vessels during the simulation, 6 times a <em>simulation day</em>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IDENTIFICATION: REQ_HMV_005</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION: The system must be able to generate past position(s) of a vessel, based on the DMPs of the vessel-type, 6 times a <em>day</em>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IDENTIFICATION: REQ_HMV_006</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION: The user must be able to request the crew/ownership, stored on an earlier moment or generated by a lack of history.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IDENTIFICATION: REQ_HMV_007</th>
</tr>
</thead>
<tbody>
<tr>
<td>PURPOSE: The have the knowledge about the nationality of the crew and the possible changes of the ownership.</td>
</tr>
<tr>
<td>DESCRIPTION: The system must be able to store the crew/ownership per vessel weekly in <em>simulation time</em>.</td>
</tr>
</tbody>
</table>
2.1.4 Diagram

A diagram will provide a better representation of the several parts, discussed above. We have cut the diagram in half and spread it over two pages in order to increase the readability.

Figure 2. Requirements Diagram (part 1)
Describing overlap and inheritance between the several abstraction parts.
Figure 3. Requirements Diagram (part 2)

*Describing overlap and inheritance between the several abstraction parts.*
2.2 Non-Functional Requirements

This chapter provides a list of requirements which are not based on the functionality of the system.

2.2.1 Usability

**Scenario developers**
The involvement of the human input is partly defined in VR Forces GUI. The scenario will be created in this GUI.

**Analyzers or C2 application users**
On the other hand, while analyzing, a user would like to obtain information about one or multiple entities or about a certain situation.

2.2.2 Reliability

The information about the way entities move are based on real life data. The DMPs are created based on the AIS information, to provide a realistic simulation of the shipping industry.
3. Scenarios & Use Cases

In this chapter we discuss the possible scenarios after that the resulting derived use cases. But before these scenarios can be made, we should take a look at the possible actors, which play a role.

3.1 Actors

We have to deal with the following type of actors. The actors are also visualized in a schematic representation in figure 1.

Users
- Role player
  *The user, who will perform influence during the simulation*
- C2 touch table users
  *The users, who wants to do research and use the system for educational/training purposes.*
- Scenario builders
  *The users, who will create a scenario, in which the simulation will take place.*

System
- Specific component of the system
  *The components of the system, which communicate with other parts of the system.*

![Figure 1. Schematic representation of the interactions of the system](image-url)
3.2 Scenarios

We have split the scenarios in two main parts. The scenarios with the interaction between the user and the system on the other hand the interaction between the several parts of the system.

3.2.1 USER – SYSTEM

The scenarios are directly derived from the requirements. Only the requirements, which imply interaction with the system, are summed as scenarios. The code, behind each scenario is the reference to the use cases, described in the next section. This code is established this way:

\[\text{User|System}-[\text{Abstraction level: General|STG|MSA}]{\text{number}}\]

Role player
1. Add entities #U-RP01
2. Change DMP of an entity #U-RP02
3. Disconnect DMPs of an entity #U-RP03
4. Store actions of the Role Player in a database #U-RP04
5. Publish an INTELL report #U-RP05
6. Manage simulation speed #U-RP06

C2
1. Request data about an entity using a select-query from a database #U-CC01
2. Request data about an a sensor using a select-query from a database #U-CC02
3. Request data about a report database using a select-query #U-CC03

Scenario builder
1. Import geographic areas, roads and points #U-SB01
2. Create area’s and assign one or multiple functions to it #U-SB02
3. Create routes and assign one or multiple functions to it #U-SB03
4. Create spawn points and assign one or multiple functions to it #U-SB04
5. Create points from where entities will be removed #U-SB05
6. Save the created scenario #U-SB06
7. Load (saved) scenario using VR Forces #U-SB07

3.2.2 SYSTEM – SYSTEM

The system-with-system scenarios are on the same way derived as the above listed scenarios, using the requirements. Here, the code is established this way:

\[\text{User|System}-[\text{Abstraction level: General|STG|MSA}]{\text{number}}\]

General, STG
1. Assign one or multiple properties to an area #S-G01
2. Assign one or multiple properties to a route #S-G02
3. Generate entities for a route #S-G03
4. Assign one or multiple properties to a spawn point #S-G04
5. Generate entities for a spawn point #S-G05
6. Create point from where entities will be removed #S-G06
7. Assign density to an area of route #S-G07
8. Store ship information in a data store #S-G08
9. Initialize a scenario based on a random start time #S-G09
10. Import geographic areas, road and points #S-S10
11. Save the created scenario #S-G11
12. Accelerate the simulation speed #S-G12
13. *The* entity generator #S-G13
14. The entity generator must have a persistent behavior #S-G14
15. The DMPs must have a persistent behavior #S-G15

**MSA**
16. Simulate vessel behavior, based on DMPs #S-M16
17. After a DMP modification, the system stores history based on the new DMP #S-M17
18. Every vessel should have a unique behavior #S-M18
19. Provide a complete log for the Sea Traffic Generator #S-M19
20. Generate specific AIS information about an entity. #S-M20
21. Keep a log of the entity-type modifications #S-M21
22. Store position(s) #S-M22
23. Generate past position(s) #S-M23
24. Store crew/ownership #S-M24
25. Store Alpha report(s) #S-M25
26. Generate Alpha reports(s) #S-M26

### 3.3 Use cases

In this chapter, the scenarios, described in the previous section will be worked out in Use Cases. Therefore we will use the same structure as in the previous section; the interaction division of user-with-system and system-with-system. Per division the scenarios will return, like we listed in the scenario section (1.2).

#### 3.3.1 USER - SYSTEM

In this section, the use cases related to the interactions between the user and the system will be elaborated. First, the input of the Role Player will be explained, followed by the possibilities of the C2 user(s). The way the scenario builder is able to work, is covered the last.

Every Use Cases contains several fields: the use case number with related title, the actor of the use case, the preconditions, the requirements\(^\text{20}\) and the flow of events between the actor and the system.

\(^{20}\) For detailed information we refer to the Requirement Analysis Document Ch.2.1
3.3.1.1 Role Player Use Cases

While reading the use cases of the Role Player it is important to keep the related defined Pre-
condition in mind. It is called “Precondition Role Player” (or PR): The possibility to influence the
simulation (using a Role player) is allowed after the simulation has been started.

<table>
<thead>
<tr>
<th>USE CASE #U-RP01</th>
<th>Add entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Role Player user</td>
</tr>
<tr>
<td>Precondition</td>
<td>PR</td>
</tr>
<tr>
<td></td>
<td>System must contain at least one DMP.</td>
</tr>
<tr>
<td></td>
<td>This Role Player must be connected with VR Forces and therefor the Role Player is situated in the VR Forces/SIMOBS component.</td>
</tr>
<tr>
<td>Requirements</td>
<td>REQ_GRP_001</td>
</tr>
<tr>
<td>Flow of events</td>
<td></td>
</tr>
<tr>
<td>Actor</td>
<td>System</td>
</tr>
<tr>
<td>1. Selects a point X on the map using a indication-device like a mouse and click add an entity</td>
<td>2. Display a menu, with possible DMPs to assign to the entity.</td>
</tr>
<tr>
<td>3. Choose DMP and click add</td>
<td>4. Add an entity with a linked DMP on the map at position X.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE CASE #U-RP02</th>
<th>Change DMP of an entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Role Player user</td>
</tr>
<tr>
<td>Precondition</td>
<td>System must contain at least one DMP.</td>
</tr>
<tr>
<td></td>
<td>This Role Player must be connected with VR Forces and therefor the Role Player is situated in the VR Forces/SIMOBS component.</td>
</tr>
<tr>
<td>Requirements</td>
<td>REQ_GRP_002</td>
</tr>
<tr>
<td>Flow of events</td>
<td></td>
</tr>
<tr>
<td>Actor</td>
<td>System</td>
</tr>
<tr>
<td>1. Select an entity, click change DMP</td>
<td>2. Show selection menu from which the entity’s DMP can be chosen</td>
</tr>
<tr>
<td>3. Choose DMP and click edit</td>
<td>4. Simulate according to his new DMP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE CASE #U-RP03</th>
<th>Disconnect DMPs of an entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Role Player user</td>
</tr>
<tr>
<td>Precondition</td>
<td>System must contain at least one DMP.</td>
</tr>
<tr>
<td></td>
<td>This Role Player must be connected with VR Forces and therefor the Role Player is situated in the VR Forces/SIMOBS component.</td>
</tr>
<tr>
<td>Requirements</td>
<td>REQ_GRP_003</td>
</tr>
<tr>
<td>Flow of events</td>
<td></td>
</tr>
<tr>
<td>Actor</td>
<td>System</td>
</tr>
<tr>
<td>1. Select an entity, click disconnect DMPs</td>
<td>2. Simulate according to VR Forces (or manually)</td>
</tr>
</tbody>
</table>

| USE CASE #U-RP04 | Log the actions of the Role Player in a database |
### Actor: Role Player user  
### Precondition: PR  
### Requirements: REQ_MRP_004

#### Flow of events

<table>
<thead>
<tr>
<th>Actor</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Perform a Role Player action</td>
<td>2. Log this action, with parameters: actual time, simulation time, action, values before and after the change.</td>
</tr>
</tbody>
</table>

---

**USE CASE #U-RP05**  
**Publish an INTELL report**

| Actor | Role Player user  
|-------|------------------|
| Precondition: PR  
This Role Player must be a part of the History Generator, in order to store reports. |
| Requirements: REQ_MRP_005

#### Flow of events

<table>
<thead>
<tr>
<th>Actor</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select entity and click <em>publish report</em> -&gt; INTELL report</td>
<td>2. Store an INTELL report linked to the selected ship.</td>
</tr>
</tbody>
</table>

---

**USE CASE #U-RP06**  
**Manage simulation speed**

| Actor | Scenario builder user  
|-------|------------------------|
| Precondition: PSB  
The simulation must be running  
This Role Player must be connected with VR Forces and therefor the Role Player is situated in the VR Forces/SIMOBS component. |
| Module: VR Forces  
Requirements: REQ_GSB_005

#### Flow of events

<table>
<thead>
<tr>
<th>Actor</th>
<th>System</th>
</tr>
</thead>
</table>
| 1. Using VR Forces, click *manage simulation speed* | 2. Show a screen with a slider to adjust the simulation speed.  
3. Choose the desired simulation speed and click *edit*  
4. The simulation will use the new simulation speed (see Use Case #S-G12) |

---

### 3.3.1.2 C2 Use Cases

Before reading the Use Cases, related to C2, we have also a pre-condition for this section: “Precondition C2” (or PC): The possibility to request data about the simulation (using C2) is allowed after the simulation has been started.

**USE CASE #U-CC01**  
**Request data about an entity using a select-query form a database**

<table>
<thead>
<tr>
<th>Actor</th>
<th>C2 user</th>
</tr>
</thead>
</table>
| Precondition: PC  
The system must contain history either generated or stored. |
| Requirements: REQ_HMV_001, REQ_HMV_003, REQ_HMV_006

#### Flow of events

<table>
<thead>
<tr>
<th>Actor</th>
<th>System</th>
</tr>
</thead>
</table>
| 1. Using VR Forces, click *manage simulation speed* | 2. Show a screen with a slider to adjust the simulation speed.  
3. Choose the desired simulation speed and click *edit*  
4. The simulation will use the new simulation speed (see Use Case #S-G12) |
1. Select one or multiple entities and click request entity information

2. Show a screen with multiple input possibilities, like the log of a ship, the positions, information about the crew/ownership and a time range.

3. Fill in the wanted input fields and click request

4. By using Use Case #S-M23, generate history of the selected entities.
5. Display a screen with a representation of the requested information

**USE CASE #U-CC02**

Request data about a sensor using a select-query form a database

<table>
<thead>
<tr>
<th>Actor</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2 user</td>
<td></td>
</tr>
</tbody>
</table>

**Precondition**

- PC
  - The system must contain data about sensors.

**Requirements**

REQ_RCS_001, REQ_EOI_001, REQ_AIS_001, REQ_PIC_001

**Definition**

Sensor: Radar Cross Section, EO/IR, AIS information, Picture information

**Flow of events**

1. Select one or multiple entities and click request sensor information

2. Show a screen with a selection of the possible sensors (see definition).

3. Select the desired sensor and click request

4. IF HISTORY ALREADY EXISTS:
   - use the existing history
   ELSE
   - using Use Case #S-M20, generate (AIS sensor) history of the selected entities.

5. Request the other sensor data using an external database with this sensor information.

6. Display a screen with a representation of the requested information

**USE CASE #U-CC03**

Request data about a report database using a select-query

<table>
<thead>
<tr>
<th>Actor</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2 user</td>
<td></td>
</tr>
</tbody>
</table>

**Precondition**

- PC
  - The system must contain data about reports stored in a database

**Requirements**

REQ_INT_001, REQ_MAR_001

**Definition**

Report: Alpha report, INTELL report

**Flow of events**

1. Select one or multiple entities and click request report information

2. Show a screen with a selection of the possible sensors (see definition).

3. Select the desired report and click request

4. Get the required information form
the selected entities and the report-type from the database.
6. Display a screen with a representation of the requested information

### 3.3.1.3 Scenario builder Use Cases

In this section, we also introduce a precondition first, called “Precondition Scenario Builder” (or PSB):

**VR Forces must be running.**

<table>
<thead>
<tr>
<th>USE CASE #U-SB01</th>
<th>Import geographic areas, roads and points</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>Scenario builder user</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>- PSB</td>
</tr>
<tr>
<td></td>
<td>- The imported file must be valid.</td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td>VR Forces</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>REQ_GSB_005</td>
</tr>
<tr>
<td><strong>Flow of events</strong></td>
<td><strong>Actor</strong></td>
</tr>
<tr>
<td></td>
<td>1. Using VR Forces, click load -&gt;</td>
</tr>
<tr>
<td></td>
<td>geometry area/roads/points</td>
</tr>
<tr>
<td></td>
<td>3. Choose the file, containing the</td>
</tr>
<tr>
<td></td>
<td>geometry area/roads/points and click</td>
</tr>
<tr>
<td></td>
<td>load</td>
</tr>
<tr>
<td></td>
<td><strong>System</strong></td>
</tr>
<tr>
<td></td>
<td>2. Show a screen with the possibility to</td>
</tr>
<tr>
<td></td>
<td>browse through your files at your pc.</td>
</tr>
<tr>
<td></td>
<td>4. Show the imported areas/roads</td>
</tr>
<tr>
<td></td>
<td>(See Use Case #S-S10).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE CASE #U-SB02</th>
<th>Create area’s and assign one or multiple functions to it</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>Scenario builder user</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>- PSB</td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td>VR Forces</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>REQ_GSB_001</td>
</tr>
<tr>
<td><strong>Flow of events</strong></td>
<td><strong>Actor</strong></td>
</tr>
<tr>
<td></td>
<td>1. Using VR Forces, click create-&gt; area’s</td>
</tr>
<tr>
<td></td>
<td>3. Create an area by clicking multiple point</td>
</tr>
<tr>
<td></td>
<td>5. Click on option -&gt; assign function</td>
</tr>
<tr>
<td><strong>System (VR Forces)</strong></td>
<td>2. A specific mouse cursor will be displayed.</td>
</tr>
<tr>
<td></td>
<td>4 Connect the first point with the second one.</td>
</tr>
<tr>
<td></td>
<td>Then connect the second one with the third one.</td>
</tr>
<tr>
<td></td>
<td>Repeat till all points of the set have been connected.</td>
</tr>
<tr>
<td></td>
<td>Mark the connected points as “area”.</td>
</tr>
<tr>
<td></td>
<td>6. Show a screen with the possibility to select one or multiple functions, that must be assigned to the created area.</td>
</tr>
</tbody>
</table>
7. Select one or multiple functions and click apply.

8. The functions are assigned to the area as described in **Use Case #S-G01**.

<table>
<thead>
<tr>
<th>USE CASE #U-SB03</th>
<th>Create routes and assign one or multiple functions to it</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>Scenario builder user</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>PSB</td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td>VR Forces</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>REQ_GSB_002</td>
</tr>
</tbody>
</table>

**Flow of events**

<table>
<thead>
<tr>
<th>Actor</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Using VR Forces, click <em>create</em>-&gt;<em>route</em></td>
<td>2. A specific mouse cursor will be displayed.</td>
</tr>
<tr>
<td>3. Create an area by clicking multiple points</td>
<td>4. Connect the first point with the second one. Then connect the second one with the third one. Repeat till all points of the set have been connected. Mark the connected points as “route”.</td>
</tr>
<tr>
<td>5. Click on <em>option</em> -&gt; assign <em>function</em></td>
<td>6. Show a screen with the possibility to select one or multiple functions, that must be assigned to the created route.</td>
</tr>
<tr>
<td>7. Select one or multiple functions and click <em>apply</em>.</td>
<td>8. The functions are assigned to the route as described in <strong>Use Case #S-G02</strong>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE CASE #U-SB04</th>
<th>Create spawn points and assign one or multiple functions to it</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>Scenario builder user</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>PSB</td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td>VR Forces</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>REQ_GSB_003</td>
</tr>
</tbody>
</table>

**Flow of events**

<table>
<thead>
<tr>
<th>Actor</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Using VR Forces, click <em>create</em>-&gt;<em>spawn points</em></td>
<td>2. A specific mouse cursor will be displayed.</td>
</tr>
<tr>
<td>3. Create an area by clicking multiple point</td>
<td>4. Connect the first point with the second one. Then connect the second one with the third one. Repeat till all points of the set have been connected. Mark the connected points as “route”.</td>
</tr>
</tbody>
</table>
### USE CASE #U-SB05

**Actor**
Scenario builder user

**Precondition**
PSB

**Module**
VR Forces

**Requirements**
REQ_GSB_00

### Flow of events

<table>
<thead>
<tr>
<th>Actor</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Using VR Forces, click <em>create-</em> &gt; <em>removal point</em></td>
<td>2. A specific mouse cursor will be displayed.</td>
</tr>
<tr>
<td>3. Click on the a location on the map where a entity will be removed. Click <em>update</em></td>
<td>4. The removal/sync points will be processed as described in Use Case #S-G06.</td>
</tr>
</tbody>
</table>

5. Click on *option* -> *assign property*

7. Select the desired functions and click *apply*.

4. The area based on the clicked points will be created.

6. Show a menu with possible functions.

8. The selected functions will be processed and be displayed in the VR Forces GUI, using VR Forces (See Use Case #S-G04).

---

**USE CASE #U-SB06**

**Actor**
Scenario builder user

**Precondition**
PSB

**Module**
VR Forces

**Requirements**
REQ_GSB_006

### Flow of events

<table>
<thead>
<tr>
<th>Actor</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Using VR Forces, click <em>save as-</em> &gt; <em>build scenario</em></td>
<td>2. Show a screen with the possibility to browse through your files at your pc. Also display selection fields of the several parts, which the user wants to save like: geometry, entities, areas, routes.</td>
</tr>
<tr>
<td>3. Select a place to save, select the parts and the file and click save</td>
<td>4. The scenario, considering the parts, will be saved. (See Use Case #S-G11).</td>
</tr>
<tr>
<td>USE CASE #U-SB07</td>
<td>Load (the) saved scenario using VR-Forces</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td><strong>Actor</strong></td>
<td>Scenario builder user</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>- PSB</td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td>VR Forces</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>REQ_GSB_005</td>
</tr>
</tbody>
</table>

**Flow of events**

<table>
<thead>
<tr>
<th><strong>Actor</strong></th>
<th><strong>System</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Using VR Forces, click <em>load -&gt; built scenario</em></td>
<td>2. Show a screen with the possibility to browse through your files at your pc. <em>Also display selection fields of the several saved parts, which the user wants to save like: geometry, areas (containing entities), routes.</em></td>
</tr>
</tbody>
</table>

3. Choose the file, select the *parts* that must be displayed and click *load*  
4. The selected scenario file will be processed and be displayed, considering the *parts*, in the VR Forces GUI, using VR Forces. (See **Use Case #S-S10**).
3.3.2 System-System

This section describes the use cases of the interactions between the components of the system. First the general and Sea Traffic Generator Use Cases are elaborated and then the Use Cases related to MSA. This division is achieved by the requirements.

3.3.2.1 General & Sea Traffic Generation

<table>
<thead>
<tr>
<th>USE CASE #S-G01</th>
<th>Assign one or multiple functions to an area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>System: VR Forces</td>
</tr>
<tr>
<td>Precondition</td>
<td>- An area must been selected or created and at least one function chosen.</td>
</tr>
<tr>
<td>Requirements</td>
<td>REQ_GEN_001</td>
</tr>
<tr>
<td>Flow of events</td>
<td></td>
</tr>
<tr>
<td>Actor</td>
<td>SIMOBS logic</td>
</tr>
<tr>
<td>1. An area has been selected/created by the user.</td>
<td>2. Assign the chosen function(s) to the area.</td>
</tr>
<tr>
<td>3. From now on the area will be used, the way these function(s) describe.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE CASE #S-G02</th>
<th>Assign one or multiple functions to a route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>System: VR Forces</td>
</tr>
<tr>
<td>Precondition</td>
<td>- An route must been selected or created and at least one function chosen.</td>
</tr>
<tr>
<td>Requirements</td>
<td>REQ_GEN_003</td>
</tr>
<tr>
<td>Flow of events</td>
<td></td>
</tr>
<tr>
<td>Actor</td>
<td>SIMOBS Logic</td>
</tr>
<tr>
<td>1. A route has been selected/created by a user.</td>
<td>2. Assign the chosen function(s) to the route.</td>
</tr>
<tr>
<td>3. From now on the route will be used the way these function(s) describe.</td>
<td></td>
</tr>
<tr>
<td>USE CASE</td>
<td>Generate entities for a route</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td><strong>Actor</strong></td>
<td>System: VR Forces</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>The scenario must contain a route.</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>REQ_GEN_004</td>
</tr>
</tbody>
</table>

**Flow of events**

<table>
<thead>
<tr>
<th>Actor</th>
<th>SIMOBS LOGIC</th>
<th>DMPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Generate the different kind of entities, based on the properties and a possible start time t.&lt;sub&gt;x&lt;/sub&gt;.</td>
<td>2. Ask for the behavior. By sending a request to DMP database. 4. Assign the behavior to generated entities. Using the DMPs, generate the start positions of the entities according to start time t.&lt;sub&gt;x&lt;/sub&gt; (see Use Case #S-G09: Initialize a scenario based on a random start time).</td>
<td>3. Return the behavior.</td>
</tr>
<tr>
<td>5. Position the entities according to their generated start position.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE CASE</th>
<th>Assign one or multiple function(s) to a spawn point</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>System: VR Forces</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>An area must been selected, valid functions chosen.</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>REQ_GEN_005</td>
</tr>
</tbody>
</table>

**Flow of events**

<table>
<thead>
<tr>
<th>Actor</th>
<th>SIMOBS Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A spawn point has been selected/created by a user. 3. From now on use the spawn point, the way these function(s) describe.</td>
<td>2. Assign the chosen function(s) to the spawn point.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE CASE</th>
<th>Generate entities for a spawn point</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>System: VR Forces</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>At least one spawn point must exist</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>REQ_GEN_006</td>
</tr>
</tbody>
</table>

**Flow of events**

| Actor | |
|-------| |
1. Generate the different kind of entities, based on the properties/functions.

<table>
<thead>
<tr>
<th>USE CASE #S-G06</th>
<th>Create points from where entities will be removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>System: VR Forces</td>
</tr>
</tbody>
</table>
| Precondition    | - The user must have clicked on 1 position in the area.  
|                 | - This Use Case only applies to ships which are generated by spawn points (Use Case #S-G05) |
| Requirements    | REQ_GEN_007                                      |
| Flow of events  |                                                   |
| Actor           | SIMOBS logic                                     |
|                 | 1. Compute the first point.                      |
|                 | 2. When a ship enters within a short (defined) range of this point, the entity will be removed (including linked DMP and history). |
|                 | 3. Update visualization in the GUI.              |

<table>
<thead>
<tr>
<th>USE CASE #S-G07</th>
<th>Assign density to an area or route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>System: VR Forces</td>
</tr>
<tr>
<td>Precondition</td>
<td>- An area or a route must been selected, a density value must be defined in a template</td>
</tr>
<tr>
<td>Requirements</td>
<td>REQ_GEN_008</td>
</tr>
<tr>
<td>Flow of events</td>
<td></td>
</tr>
<tr>
<td>Actor</td>
<td>SIMOBS logic</td>
</tr>
<tr>
<td></td>
<td>1. Compute the selected area/route.</td>
</tr>
<tr>
<td></td>
<td>2. Get the density-value using the template that assigned the area/route.</td>
</tr>
<tr>
<td></td>
<td>3. Apply the values of the density to the entities.</td>
</tr>
<tr>
<td></td>
<td>4. Show the chances, made after changing the density.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE CASE #S-G08</th>
<th>Store ship information in a data store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>System: AIS Generator</td>
</tr>
<tr>
<td>Precondition</td>
<td>- The simulation is running OR</td>
</tr>
<tr>
<td></td>
<td>- The C2 user must requested information</td>
</tr>
<tr>
<td>Requirements</td>
<td>GEN_009</td>
</tr>
<tr>
<td>Flow of events</td>
<td>Situation 1: The simulation is running</td>
</tr>
<tr>
<td>Actor</td>
<td>Entity Information DB</td>
</tr>
</tbody>
</table>


1. Every x times a day/week, get information about the entities and store.

2. Store this data in the database

**Situation 2: The (C2) user is requesting information**

<table>
<thead>
<tr>
<th>Actor</th>
<th>Entity Information DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Check if the required history is complete (number of days stored)</td>
<td>2. Check: is required history complete</td>
</tr>
<tr>
<td>3a. NO: generate history, in order to make the history complete (30 days minus the history, that have been stored already), using <strong>Use Case #S-M20</strong>: Generate specific AIS information about an entity.</td>
<td>3b. YES: do nothing</td>
</tr>
<tr>
<td>4a. Store the generated history, to complete the set of history</td>
<td></td>
</tr>
</tbody>
</table>

**USE CASE #S-G09**

<table>
<thead>
<tr>
<th>Initialize a scenario based on a random start time ( t_x )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
</tr>
</tbody>
</table>

**Flow of events**

<table>
<thead>
<tr>
<th>Actor</th>
<th>SIMOBS logic</th>
<th>AIS Generator</th>
<th>DMPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Determine the current AIS of the entities</td>
<td>2. AIS Generator to generate current position of the entities at time ( t_x )</td>
<td>3. Generate the current positions of the entities. In order to do this, calculate the positions using DMPs.</td>
<td>4. Use the finite state machines of the DMP</td>
</tr>
</tbody>
</table>

**USE CASE #S-S10**

<table>
<thead>
<tr>
<th>Import geographic areas, road and points</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
</tbody>
</table>
**Requirements** | STG_001
--- | ---
**Flow of events**<br>**Actor**<br>1. Selected files will be processed by the SIM<br>3. Visualize the update(s)<br>**VR Forces SIMULATOR**<br>2. Update in the scenario

**USE CASE G11**<br>Save the created scenario<br>**Actor**<br>System: VR Forces<br>**Precondition**<br>- Scenario with geographic/selected areas, selected routes, assigned properties and/or entities must be inserted.<br>**Requirements** | GEN_013
--- | ---
**Flow of events**<br>**Actor**<br>1. Process all properties of the current geographic/selected areas, selected routes, assigned properties, and inserted entities<br>**Scenario DB**<br>2. Store these properties in separate database tables of the Scenario DB.

**USE CASE G12**<br>Accelerate the simulation speed<br>**Actor**<br>System: VR Forces<br>**Precondition**<br>- Simulation speed-value selected by user.<br>**Requirements** | STG_003
--- | ---
**Flow of events**<br>**Actor**<br>1. Based on the value of the simulation speed-slider, adjust the simulation speed with SIMOBS<br>**SIMOBS**<br>2. Using this value, increase the speed the simulation-loop is passed through.

**USE CASE G13**<br>The entity generator<br>**Actor**<br>System: VR Forces<br>**Precondition**<br>- DMPs available<br>**Requirements** | GNT_001
--- | ---
**Flow of events**<br>**Actor**<br>**SIMOBS logic**<br>**DMPs / Roleplayer**
1. Use SIMOBS logic, in order to generate entities

2. Has to give commands to other systems to provide the density(3) of entities in the area, distribution(3) of entity-types, the variance(3) in generating time, the concept of entity being at location X at time Y using DMP Z (4).
3. Get requested information about the density, distribution and variance using the belonging template

4. Use the DMP Z, to determine it is route to destination X at time Y.

5. Based on this information(density and distribution) and a calculation of the variance with a random generator and the route its now possible to generate one or multiple entities with a certain behavior.

6. Visualize in GUI.

**USE CASE #5-G14**  
The entity generator must have a varying output

<table>
<thead>
<tr>
<th><strong>Actor</strong></th>
<th>System: VR Forces</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precondition</strong></td>
<td>Degrees of freedom for DMPs implemented</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>GNT_002</td>
</tr>
</tbody>
</table>

**Flow of events**

- **Actor**
  1. Check whether the entity wants to use a random seed. If true then make use of 2.a. Else, use 2.b

- **SIMOBS**
  2. Using the variables of freedom for entity generation, generate entities
  3. Using default variables of the DMPs, generate entities

**USE CASE #5-G15**  
The DMPs must have a persistent behavior, with or without random seed

<table>
<thead>
<tr>
<th><strong>Actor</strong></th>
<th>System: VR Forces</th>
</tr>
</thead>
</table>
| **Precondition** | DMPs available
- Degrees of freedom for DMPs implemented |
| **Requirements** | GNT_003 |

**Flow of events**

- **Actor**

- **SIMOBS**
  2. Using the variables of freedom for entity generation, generate entities
  3. Using default variables of the DMPs, generate entities

- **DMPs**
1. Let SIMOBS take control over the entities in VR Forces.

2. Use DMP to simulate behavior.

3. Check whether the entity wants to use a random seed. If true then make use of the variables of freedom for DMPs. Else, only use the standard behavior, based on this DMP.

### MSA

<table>
<thead>
<tr>
<th>USE CASE #5-M16</th>
<th>Simulate vessel behavior, based on DMPs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>System: VR Forces</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>DMPs available</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>GDP_001, SDP_001, MDP_001</td>
</tr>
</tbody>
</table>

**Flow of events**

<table>
<thead>
<tr>
<th><strong>Actor</strong></th>
<th><strong>SIMOBS</strong></th>
<th><strong>DMPs</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Let SIMOBS take control over the simulation of the entities in VR Forces.</td>
<td>2. Use DMP to simulate behavior.</td>
<td>3. Check based on the state of the machine, which implements the behavior of the entity, and its properties/limitations, what the current/next action/position of the entity is.</td>
</tr>
<tr>
<td>5. Visualize in GUI.</td>
<td>4. Based on this action, simulate its movements.</td>
<td></td>
</tr>
</tbody>
</table>
### USE CASE #S-M17

After a DMP modification, the system stores history based on the new DMP

**Actor**
System: VR Forces

**Precondition**
- A DMP must be modified (Use Case #U-RP02: Change DMP of an entity)

**Requirements**
MDP_002

**Flow of events**

<table>
<thead>
<tr>
<th>Actor</th>
<th>SIMOBS Logic</th>
<th>DMP</th>
<th>Entity information system</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Get the entity info of the modified vessel.</td>
<td>2. Request for the modified DMP.</td>
<td>3. Return the modified DMP.</td>
<td></td>
</tr>
<tr>
<td>4. Change the behavior of the vessel according to the modified DMP.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Send an event to Entity information system to generate new history, based on the modified DMP.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6. From now on the stored ship information (see Use Case #S-G08: Store ship information in a data store) will be based on the new DMP.</td>
<td></td>
</tr>
</tbody>
</table>

### USE CASE #S-M18

Every vessel should have a unique behavior.

**Actor**
System: SIMOBS logic

**Precondition**
- The simulation must contain vessels, with linked DMPs.

**Requirements**
MDP_004

**Flow of events**

<table>
<thead>
<tr>
<th>Actor</th>
<th>DMPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Get linked DMP from database.</td>
<td>2. Provide DMP to SIMOBS.</td>
</tr>
<tr>
<td>3. Use the DMP and degrees of freedom to simulate unique behavior.</td>
<td></td>
</tr>
</tbody>
</table>

Linked to Use Case #S-G15: The DMP must have a persistent behavior, with or without random seed

### USE CASE #S-M19

Provide a complete log for the Sea Traffic Generator

**Actor**
System: C2 application

**Precondition**
- History must be stored or generated
### Requirements

**MHI_001**

#### Flow of events

<table>
<thead>
<tr>
<th>Actor</th>
<th>VR Forces</th>
<th>INTELL Database</th>
<th>Entity information DB</th>
<th>Entity information system</th>
</tr>
</thead>
</table>
| 1. Request all entities.  
2. Request logged information about entities.  
3. If DMP-changes occurred: request the log of **Use Case #5-M21**: *Keep a log of the entity-type modifications.* | 4. Provide the information about the entity. (position, course, speed, cargo,..) | 5. Return INTELL Reports | 6. Return history of vessels | 7. Make a log of the events of sea traffic, sorted on ascending simulation time. |
<p>| 8. Show the created log. | | | | |</p>
<table>
<thead>
<tr>
<th><strong>USE CASE #5-M20</strong></th>
<th>Generate specific AIS information about an entity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>System: AIS Generator</td>
</tr>
</tbody>
</table>
| **Precondition**    | - The entity does not contain sufficient (AIS) history  
                       - The C2 user must requested information about this entity |
| **Requirements**    | AIS_002                                          |

### Flow of events

<table>
<thead>
<tr>
<th>Actor</th>
<th>SIMOBS logic</th>
<th>DMP</th>
<th>AIS generator</th>
<th>Entity information DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Ask SIMOBS to generate history based on the start time $t_x$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Use AIS Generator to backtrack AIS data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Generate (30 minus the already stored days) days of history (before $t_x$), using backtracking of the DMPs(4) related to the entities and save it to the database(5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. backtrack data using the finite state machines of the DMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Save the history related to each vessel in the database</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### USE CASE #S-M21
Keep a log of the entity-type modifications

**Actor**
SIMOBS logic

**Precondition**
- An entity-type has been changed

**Requirements**
HMV_002

**Flow of events**

<table>
<thead>
<tr>
<th>Actor</th>
<th>VR Forces</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. When an entity has been changed</td>
<td>2. Insert/update log, using the previous and new DMPs with related simulation time.</td>
</tr>
</tbody>
</table>

### USE CASE #S-M22
Store (current) positions of the ship

**Actor**
SIMOBS logic

**Precondition**
- The system is currently in a simulation state

**Requirements**
HMV_004

**Flow of events**

<table>
<thead>
<tr>
<th>Actor</th>
<th>VR Forces</th>
<th>Entity Information DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. While simulating, sent every x times a day an event to receive the current position of the ship.</td>
<td>2. Determine position of the ship.</td>
<td></td>
</tr>
<tr>
<td>3. Update the database, by sending an event to the database.</td>
<td>4. Insert/update database, using the position with related simulation time.</td>
<td></td>
</tr>
</tbody>
</table>

### USE CASE #S-M23
Generate (past) positions of the ships

**Actor**
SIMOBS logic

**Precondition**
- The system is currently generating history
- The history is not complete, the number of stored days (or simulation-days) are not equal to 30

**Requirements**
HMV_005

**Flow of events**

<table>
<thead>
<tr>
<th>Actor</th>
<th>AIS generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Use the AIS history generator to generate history of a ship</td>
<td>2. Use the backtracking-principle, to generate (using the related DMP) six times a day an update of the positions with related simulation time.</td>
</tr>
</tbody>
</table>
### USE CASE #S-M24
**Store crew/ownership, weekly**

**Actor**
System: SIMOBS logic

**Precondition**
The system is currently in a simulation state

**Requirements**
HMV_007

<table>
<thead>
<tr>
<th>Flow of events</th>
<th>Actor</th>
<th>Extern database</th>
<th>Entity Information DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. While generating, check every week the crew/ownership of a vessel using an external database</td>
<td></td>
<td>2. Determine the crew/ownership of the ship</td>
<td>3. Insert/update database, using the crew/ownership with related <em>simulation time</em>.</td>
</tr>
</tbody>
</table>

### USE CASE #S-M25
**Store Alpha report(s)**

**Actor**
System: AIS generator

**Precondition**
The system is currently in a simulation state

**Requirements**
MAR_001

<table>
<thead>
<tr>
<th>Flow of events</th>
<th>Actor</th>
<th>Report DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. If a new Alpha report is generated (see Use Case #S-M26), store in database</td>
<td></td>
<td>2. Insert report to database, with a link to the concerning vessel</td>
</tr>
</tbody>
</table>

### USE CASE #S-M26
**Generate Alpha report(s)**

**Actor**
System: AIS generator

**Precondition**
The system is currently in a simulation state

**Requirements**
MAR_003

<table>
<thead>
<tr>
<th>Flow of events</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. While simulating, when a vessel is crossing an secured lane, generate every x time a day a alpha report, according to the vessel.</td>
<td></td>
</tr>
</tbody>
</table>
Appendix E: Architectural Design Document

Architectural Design Document
# Table of Contents

1. Introduction .......................................................................................................................... 146  
   1.1 Scope of the system........................................................................................................... 146  
   1.2 Document overview......................................................................................................... 146  
   1.3 Design priorities .............................................................................................................. 146  

2. System overview .................................................................................................................. 147  
   2.1 MSA block diagram ......................................................................................................... 147  
   2.2 Design pattern .................................................................................................................. 147  

3. Management ....................................................................................................................... 149  
   3.1 Data management ............................................................................................................. 149  
   3.2 Task management ............................................................................................................ 151  
   3.3 Time management ............................................................................................................ 151  

4. Boundary conditions .......................................................................................................... 154  
   4.1 Initializing ......................................................................................................................... 154  
   4.2 Terminating ...................................................................................................................... 154  
   4.3 Crashing ......................................................................................................................... 154  

5. Access and safety ................................................................................................................. 155  

6. System architecture ............................................................................................................. 156  
   6.1 General ............................................................................................................................ 156  
   6.2 State diagram .................................................................................................................. 156  
   6.3 Activity diagrams ........................................................................................................... 157  
   6.4 Class diagram ................................................................................................................ 161  
   6.5 Sequence diagrams ......................................................................................................... 164
1. Introduction

The purpose of this report is to describe the system design of the Entity Information System. Before continuing with the implementation of the system it is necessary to design the architecture for a better and clearer view of the system. To describe the system we will divide the system into various components to understand how the system clearly works. The architecture described in this document is based on the requirements described in the Requirement Analysis Document and the MoSCoW Document.

1.1 Scope of the system

We are developing an Entity Information System, which generates a realistic representation of sea traffic and compute/store the history of the vessels according to their behavior. The result of the generated sea traffic will be displayed using VR-Forces, and this simulation can be used for observing and analyzing the behaviors of vessels. For example, to detect suspicious behavior.

1.2 Document overview

Chapter 2 describes the system overview, where we explain the design pattern and decisions of our system. Then we talk about the how we handle task, time and data management in Chapter 3. According to the design described in Chapter 2 and 3 we give some boundary conditions of the system, which are described in Chapter 4. In Chapter 5 we take look at the accessibility and safety of the system. Based on design and explanations a detailed architecture of the system is can be found in Chapter 6, which contains Unified Modeling Language (UML) diagrams, like state diagram, activity diagram, class diagrams to describe relations and flows of the system.

1.3 Design priorities

The architecture of the system should be reusable for other projects. With other words we should implement the system on a certain way that the Entity Information System can be used for other projects. For example for air traffic instead of sea traffic.
2. System overview

The systems is built up from several components\(^{21}\), like VR-Forces, entity information generator, SIM Logic, C2, 3D viewer. We are responsible for developing the entity information generator component. First we briefly explain the MSA block diagram, and then we explain which design pattern we use for our system. Subsequently we discuss our design decisions regarding to the architecture.

2.1 MSA block diagram

This section describes the MSA block diagram\(^{22}\), which is created by our supervisor. In the block diagram it can be seen that the system can be divided into three main components. The first component is entity information generator, the second one is C2-module, and the third one is VR-Forces. All these components communicate with each other through HLA Evolved\(^ {23}\).

Each of the three components has human role players who can intervene the simulation at any moment.\(^ {24}\) VR-Forces have two role players, namely a scenario creator and a role player. The scenario builder can modify the scenario of the simulation and the role player can modify the behavior of an entity. The entity information generator have only one role player, who is responsible from publishing reports. The C2-module has only one role player, namely the user, who can request information about an entity.

2.2 Design pattern

We are using the principles of the Model-View-Controller (MVC) design pattern\(^ {25}\). This design pattern divides the application into three units with different responsibilities: data model (model), data presentation (view) and application logic (controller). Separating these responsibilities improves readability and reusability of code.

First we created a draft version of the architecture based on the MCV design pattern. Figure 2.1 shows the results. In our case the viewer is not within the part we develop. The viewer is in the VR-Forces component, which contains the userinterface elements.

For the controller unit we created a historyController class which is responsible for application control. With other words the history controller processes and responds to events, which are usually the result of actions of the user.

For the model unit, we first created a class named generatorModel. Which defines the incoming information, by adding logic and data relationships. After defining the information, it can be saved

---

\(^{21}\) More information about several components can be found in our Orientation Report Section 4

\(^{22}\) The MSA block diagram can be found in the Appendix A.

\(^{23}\) HLA stands for high level architecture, which works as a network server. Every component can publish information/updates on HLA, and each component can subscribe for the wanted information.

\(^{24}\) The actions and responsibilities of each role player is described in our Requirement Analysis Document Section 3.

\(^{25}\) More information about MCV can be found at http://nl.wikipedia.org/wiki/Model-view-controller-model
using a persistent storage medium, like a database. But it is also possible to use the defined information to perform several actions like generating messages.

Figure 2.1: Basic architecture of our system.
3. Management

In this chapter we discuss the management of the system. We explain how we deal with data, task and time management. These three subjects are closely related to each other and are very important for our system to work well. In the next sections we explain each of the three subjects.

3.1 Data management

Storing data is very important in our system, because it ensures that the system does not need to be set every time and that models can be easily imported. In this section we explain how we handle storing data. For the initialization, the system needs some XML files and templates to create a scenario. The machine, where the server is running on requires physical storage space to store all templates, scenarios, and models in XML format.

For our part of the implementation the system needs to store (generated) information about an entity in a database. The purpose of this storage is to provide data about an entity if someone requests for it. Figure 3.4 shows the architecture of our system, each table has a unique key, which can be used to link to another table. Bold attributes are the primary keys, while underlined attributes stands for secondary keys (these are used to link to another table.).

There are two main tables, message123 and message5, each of these tables contain information about a specific type of AIS message. These tables are used to save all AIS messages which are generated. Another table is vessel_apppearance, which contains information about the appearance of a vessel. This table is used to determine in which interval a vessel is present. According to the interval we can generate the missing AIS message for an interval, if C2 requests for it. There are also some tables used to store features set of an entity with a specific type.

All of the stored data in the database can for example be used for research or analysis purposes.

<table>
<thead>
<tr>
<th>Table</th>
<th>Table position_fixing_devices_types</th>
</tr>
</thead>
<tbody>
<tr>
<td>features_cruise</td>
<td>DEVICE_ID</td>
</tr>
<tr>
<td>feature_history</td>
<td>0</td>
</tr>
<tr>
<td>feature_per_ship_type</td>
<td>1</td>
</tr>
<tr>
<td>feature_tables</td>
<td>2</td>
</tr>
<tr>
<td>message5</td>
<td>3</td>
</tr>
<tr>
<td>message123</td>
<td>4</td>
</tr>
<tr>
<td>position_fixing_devices_types</td>
<td>5</td>
</tr>
<tr>
<td>static_vessel_info</td>
<td>6</td>
</tr>
<tr>
<td>vessel_appearance_mes5</td>
<td>7</td>
</tr>
<tr>
<td>vessel_appearance_mes123</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2: All the tables of our database.  
Figure 3.3: Table position_fixing_devices_types with data
<table>
<thead>
<tr>
<th>Table features_cruise</th>
<th>Table entity</th>
<th>Table feature_history</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEATURE_TYPE_ID</td>
<td>TableID</td>
<td>FEATURE_HISTORY_ID</td>
</tr>
<tr>
<td>FEATURE</td>
<td>EntityID</td>
<td>ENTITY_ID</td>
</tr>
<tr>
<td>FEATURE_ID</td>
<td></td>
<td>TIME_IN_SEC</td>
</tr>
<tr>
<td>VALUE</td>
<td></td>
<td>CARGO_TYPE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table message5</th>
<th>Table message123</th>
<th>Table feature_tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESSAGES5_ID</td>
<td>MESSAGE123_ID</td>
<td>TABLE_ID</td>
</tr>
<tr>
<td>ENTITY_ID</td>
<td>ENTITY_ID</td>
<td>TABLE_NAME</td>
</tr>
<tr>
<td>TIME_STAMP</td>
<td>MESSAGE_ID</td>
<td></td>
</tr>
<tr>
<td>MESSAGE_ID</td>
<td>REPEAT_INDICATOR</td>
<td></td>
</tr>
<tr>
<td>REPEAT_INDICATOR</td>
<td>USER_ID</td>
<td></td>
</tr>
<tr>
<td>USER_ID</td>
<td>NAVIGATION_STATUS</td>
<td></td>
</tr>
<tr>
<td>SPARE</td>
<td>ROT</td>
<td></td>
</tr>
<tr>
<td>AIS_VERSION_IND</td>
<td>SOG</td>
<td></td>
</tr>
<tr>
<td>IMO_NUMBER</td>
<td>POSITION_ACCURACY</td>
<td></td>
</tr>
<tr>
<td>CALL_SIGN</td>
<td>LONGITUDE</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>LATITUDE</td>
<td></td>
</tr>
<tr>
<td>CARGO_TYPE</td>
<td>COG</td>
<td></td>
</tr>
<tr>
<td>DIMENSION_TO_BOW</td>
<td>TRUE_HEADING</td>
<td></td>
</tr>
<tr>
<td>DIMENSION_TO_STERN</td>
<td>TIME_STAMP</td>
<td></td>
</tr>
<tr>
<td>DIMENSION_TO_PORT</td>
<td>RESERVED_FOR_REGIONAL_APP</td>
<td></td>
</tr>
<tr>
<td>DIMENSION_TO_STARBOARD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSFIXDEV_TYPE</td>
<td>RAIM_FLAG</td>
<td></td>
</tr>
<tr>
<td>ETA</td>
<td>COMMUNICATION_STATE</td>
<td></td>
</tr>
<tr>
<td>MAX_STATIC_DRAUGHT</td>
<td>SPARE</td>
<td></td>
</tr>
<tr>
<td>DESTINATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DTE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table feature_per_ship_type</th>
<th>Table vessel_appearance_mes5</th>
<th>Table vessel_appearance_mes123</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEATURE_ID</td>
<td>ID</td>
<td>ID</td>
</tr>
<tr>
<td>SHIP_TYPE</td>
<td>USER_ID</td>
<td>USER_ID</td>
</tr>
<tr>
<td>FEATURE</td>
<td>APPEARANCE</td>
<td>APPEARANCE</td>
</tr>
<tr>
<td>INTERVAL_MIN</td>
<td>TIME_IN_SEC</td>
<td>TIME_IN_SEC</td>
</tr>
<tr>
<td>INTERVAL_MAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALUE_IN_SET</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.4: Database architecture
3.2 Task management

Task management also plays an important role in our system. Since it is necessary to manage all incoming tasks from several inputs. For task management we are using a class `deltalist`, which is made by our supervisor. This include contains a task management structure based on delta time management.

Delta T is able to handle the task list sequential, so no common problems based on sequence will appear. Furthermore, it is able to create your own set of tasks that will be executed at the appropriate time. The task list contains the tasks ordered by ascending execution time and after a new task is added, this task will takes its place in the list, before the tasks that needs to be executed after this task and after the task that needs to be executed after this task.

The structure works quit simple. Every delta time step, you check the head of the list if one or multiple tasks need to be executed. It executes and delete the task after that. If nothing needs to be executed, update the head by reducing the delta time from its own delta. In the next section we will shown an example of task management combined with time management.

3.3 Time management

Time management also plays an important role in our system. Because time management makes it possible to let components work on the same time. With other words, if component A send an event to component B, it would be desirable that component B immediately, or with a small deviation of time, the event and reacts to it. If this is not the case, the component B, might receive the event much later than desired, which leads to a delay. With this delay the system cannot operate properly. For time management we are using HLA Evolved. This was not our decision, because the use of HLA Evolved was determined by TNO.

In the next tables, we show an example of the cooperation between the task manager and the time manager. Assume we have the following interval (\( X_0, X_n \)). Where \( X_0 \) stand for the start time and \( X_n \) stands for the stop time. The system will be running the interval with a time step of 1.0.

For example let \( X_0 = 0.0 \) and \( X_n = 10.0 \). Then the system will pass the interval within the following points: 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 and ending in 10.0

<table>
<thead>
<tr>
<th>( X_0 )</th>
<th>( X_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>10.0</td>
</tr>
</tbody>
</table>

Table 3.3.1: the general interval (\( X_0, X_n \)).

<table>
<thead>
<tr>
<th>( X_0 )</th>
<th>( X_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>10.0</td>
</tr>
</tbody>
</table>

Table 3.3.2: the general interval (0.0, 10.0) with delta time =1.
In this case the difference between each step is 1.0, which is also called the delta time. This leads to the following general case:

![Table 3.3.3: the general interval \((X_0, X_n)\) with delta step = \((X_n - X_{n-1})\).](image)

Assume that the system needs to executed some tasks. Table 3.3 shows the list containing all tasks that need to be executed by the system. We just simplify the task specification with task# followed by the entity ID to which the task belong.

<table>
<thead>
<tr>
<th>Task</th>
<th>Entity ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>6</td>
</tr>
<tr>
<td>Task 2</td>
<td>6</td>
</tr>
<tr>
<td>Task 3</td>
<td>4</td>
</tr>
<tr>
<td>Task 4</td>
<td>999</td>
</tr>
<tr>
<td>Task 5</td>
<td>5</td>
</tr>
<tr>
<td>Task 6</td>
<td>999</td>
</tr>
<tr>
<td>Task 7</td>
<td>4</td>
</tr>
<tr>
<td>Task 8</td>
<td>5</td>
</tr>
<tr>
<td>Task 8</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 3.3.4: list with tasks that needs to be executed by the system

Combining table 3.3.2 with table 3.3.4, gives the table 3.3.5. Where we added the task list to time step 1. The first task will be executed immediately, because delta time is 0 and this is smaller than the time step. The delta time of the next task will be updated regarding to the current delta time. Then the system continues with checking the second task. For this task it is also time to be executed, since delta time is also 0, which is also smaller than the time step. The delta time of the next task will be updated and the system will check the next task. The third task, which has a delta time of 0.9. This also smaller than the time step, so this task will also be executed. Update the delta time of the next task, regarding to the current delta time. So the delta time of task 4 changes to 0.90, this is also smaller than time step 1. Therefore task 4 will also be executed. Repeat the same procedure for task 5. Update the delta time of task 5, this is still smaller than the time step. So this task will be executed too. Going to the next task, update delta time. Task 6 then has a delta time = 8.7, this is bigger than the time step, so this task will not be executed yet. Since the delta time is bigger than the time step, the system moves on to the next time step and updates the delta time of the task above in the list, which now is task 6. Delta time of this task will be update by reducing it with the time step. This continues until the delta time of task 6 is smaller than the time step. In time step 9 the delta time is finally smaller than the time step, so this task will be executed. This procedure continues on until all task have been executed and the task list is empty.
<table>
<thead>
<tr>
<th>#1 task 1 eld 6 uitgevoerd</th>
<th>#2 task 2 eld 6 uitgevoerd</th>
<th>#3 task 1 eld 4 uitgevoerd</th>
<th>#4 task 1 eld 999 uitgevoerd</th>
<th>#5 task 1 eld 6 uitgevoerd</th>
<th>#6 voertask niet uit, verlagen met timestep</th>
</tr>
</thead>
<tbody>
<tr>
<td>dT: 0 entId: 6</td>
<td>dT: 0 entId: 6</td>
<td>dT: 0 entId: 6</td>
<td>dT: 0 entId: 6</td>
<td>dT: 0 entId: 6</td>
<td>dT: 0 entId: 6</td>
</tr>
<tr>
<td>dT: 0.9 entId: 4</td>
<td>dT: 0.9 entId: 4</td>
<td>dT: 0 entId: 5</td>
<td>dT: 0 entId: 5</td>
<td>dT: 0.9 entId: 5</td>
<td>dT: 0.9 entId: 5</td>
</tr>
<tr>
<td>dT: 7.8 entId: 999</td>
<td>dT: 7.8 entId: 999</td>
<td>dT: 7.8 entId: 999</td>
<td>dT: 7.8 entId: 999</td>
<td>dT: 7.8 entId: 999</td>
<td>dT: 7.8 entId: 999</td>
</tr>
<tr>
<td>dT: 10.1 entId: 5</td>
<td>dT: 10.1 entId: 5</td>
<td>dT: 7.8 entId: 999</td>
<td>dT: 7.8 entId: 999</td>
<td>dT: 11.1 entId: 4</td>
<td>dT: 11.1 entId: 4</td>
</tr>
<tr>
<td>dT: 10.1 entId: 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.3.5: Interval with task list.**

### Diagram:

- **Time step 1:**
  - Task 1
    - dT: 0.7 entId: 999
- **Time step 2:**
  - Task 1
    - dT: 6.7 entId: 999
  - Task 2
    - dT: 1.1 entId: 4
- **Time step 3:**
  - Task 1
    - dT: 1.2 entId: 999
  - Task 2
    - dT: 0 entId: 4
- **Time step 9:**
  - Task 1
    - dT: 0.8 entId: 4
  - Task 2
    - dT: 0 entId: 4

### Notes:

- Each timestep represents the execution of a task with specific durations and event IDs.
- The chart illustrates the progression of tasks with intervals between them.
- The tasks are numbered sequentially from #1 to #6, indicating their order of execution or priority.
- The intervals and task lists are indicated with明明 precision, showing the timeline of activity for each task.

---

*Thesis Bachelor Project: Sea Traffic Generator*
4. Boundary conditions

The section describes how the system should handle in certain cases like, initializing, terminating, and crashing caused by extern factors.

4.1 Initializing

When the system is initializing, all templates, models, scenarios, geographic areas should be created/loaded. All components should start and a connection between C2 and the entity information generator should be created. Additionally, HLA should be initialized. Moreover, a connection between the database handler and database should also be set up when the simulation is initializing.

4.2 Terminating

At termination of the system, all components (like VR-Forces, C2 application) end and the temporary files are discarded. Unless the player gives a command to save the data or scenario. Moreover, all connections between components and databases will be broken and the database should be cleaned up.

4.3 Crashing

When the system crashes, all components should stop working and the system must give a notification to the user that the system has crashed. All files related to the simulation should be saved temporarily, so the user can reload the scenario after the crash.
5. Access and safety

To ensure that nothing goes wrong with the security of the system we need to define the rights that each role player can have. Because each modification of an entity or template can change the behavior of a entity, and since we want to create a realistic view of sea traffic, we do not want to have unreliable behaviors occur.

We categories three type of roleplayers\(^{26}\). The first roleplayer is the user behind the C2 application, this user only has the right to request for information, he/she cannot modify any of the data. So this user only has the right to access the system on a high level. Second type of roleplayer is the one inside the entity information generator. This user can only publish INTELL reports, he/she does not have the right to access other information then the INTELL reports. Third types of roleplayers are scenariobuilder and roleplayer in VR-Forces. The scenario builder only has the right to build or modify a scenario; he/she does not have the right to access information from the database. The roleplayer in VR-Forces has the right to access all data and modify each entity in the system. This player has almost the full rights to access every part of the system, except the implementation code.

Detailed descriptions about the actions performed by the several roleplayers are described in Section 3 of our Requirement Analysis Document.

In principle only the above mentioned roleplayers have global access to the system. Other player can access the system by getting permission from our supervisor. The system consists of several components that interact with each other. Every single component can publish information/updates on HLA. Every component who is interested in the published data can subscribe to it, so the subscribed component will only receive data that is interesting for that component. With other words every component that is linked to HLA can have access to the published information/data/updates if it has subscribed to these information/data/updates.

\(^{26}\) See Requirement Analysis Document section 3 Scenarios for more information about the several role players.
6. System architecture

In this chapter we discuss the architecture of our system. We use UML diagrams to clearly describe the classes and relations between each class. First we give a high level description of how the system works. Second we use a state diagram to explain the simulation of Sea traffic Generator. Third we explain the activities of various components of the system. Followed by a class diagram which states all classes and relations between components. Finally we explain the communication and interaction between all classes by making use of sequence diagrams.

6.1 General

The simulation starts in VR-Forces, which creates entities and scenarios. Ground truth entity information will be published on HLA and entity information generator will receive requests/updates about entities. With these requests/updates the historyContoller can ask the generator to perform actions like generating (specific) AIS information. This step will be repeated once a time, (e.g. 5 minutes). Every time the historyContoller receives an updates, callback or request, it should handles these updates and perform an action. This continues until the simulation ends. See the next section for a high level state diagram, describing the simulation process.

6.2 State diagram

This is general state diagram of the simulation, which describes basic the flow of the system.

First a scenario will be built. Then all entities will be initialized. Then the simulation really starts. During the simulation the system can received different kind of callback to perform a certain action. The system could be paused lading in the idle state. When the system terminates it ends in the final state.

Figure 6.2: State diagram of system. Created on February 18
6.3 Activity diagrams

According to the state diagram we developed some activity diagrams which explain the possible activities in the system. We will not explain the activity diagram in details, because the figures say enough. These diagrams have been updated till March 03, 2012. After that we have not updated it anymore, because we were focusing on the milestones of the implementation.

VR FORCES  
Input: user | Output: HLA

[Diagram of activity diagram of VR Forces]

Figure 6.3.1: Activity diagram of VR Forces.
Figure 6.3.2: Activity diagram of role player in VR Forces.
Figure 6.3.3: Activity diagram of the first thoughts of the history controller.
Role Player (in history generator)  

Input: user  
Output: database update

is simulating?  
[ no ]  

Select one or multiple entities

Select INTELL Report of the entities

Store Role Player-action  
Store INTELL report in dB

Figure 6.3.4: Activity diagram of the first thought of a role player in the history controller.
6.4 Class diagram

The MSA block diagram forms the basis for our design of the entity information generator. It should be possible to generate entity information and save the information in the database. Based on this fact and the requirements described in the Requirement Analysis Document we designed the following system. The figure below describes the class structure of the architecture of our system.

We decided to make the generator class abstract, because there are one or more generators can be used to perform a specific action, and in the architecture the generator should not know which generators are available. Since the generator is only responsible for pushing tasks to the generators.

Figure 6.4 shows the class diagram regarding to the final delivered product. Due the amount of space the class diagram has been cut in two parts. Note that the Misc package does not have any arrows since these classes used by most of the other classes; we have decided not to place any arrows, to prevent an unclear class diagram. The HLAfederate class forms the bridge between HLA and the controller. It contains the callback functions. The HistoryController, can be seen as a recognizor of the system. It recognizes the incoming callbacks and couples a certain action to it. Then the controller sends the data to the Model who adds logic to it and passes the task to the generator who executes the task with a specific generator. After performing a task the data could be stored in the database, which could also return the data to the controller. There are different kind of tasks that could be performed. By performing a task AIS object could be made.
Figure 6.4a: Part one of the complete class diagram.
Figure 6.4b: Part one of the complete class diagram.
6.5 **Sequence diagrams**

This section provides the communication between the several classes using a sequence diagram. We start with the sequences where callbacks are coming in from HLA, and then we continue with the sequence of our implementation. These sequences have been updated till the first submission to SIG.

---

**Figure 6.5.0: Callback sequence**

**Callback**

```
|
+-----------------+------------+---------------------+
| federate: Hisfederate | hlaWorld: EOM | mHC : HistoryController |
|                   |             |                      |
+-----------------+------------+---------------------+
|                 |             | Create radiotransmitter |
|                 |             |                      |
| ship has marking |             |                      |
+-----------------+------------+---------------------+
| alt1            |             |                      |
|                 |             | CONTINUE WITH SEQUENCE HandleNewEntityCallback see figure below |
|                 |             |                      |
| alt 2           |             |                      |
|                 |             | CONTINUE WITH SEQUENCE HandleEntityDeleteCallback see figure below |
```

**HandleNewEntityCallback**

```
|
+-----------------+------------+---------------------+
| mHC : HistoryController | appTask : AppearanceTask |
| handleNewEntityCallback(entityID, EInputType(0)) | create appearanceTask |
| create appTask(NULL, entityID, true, time (NULL)) |
| addTask(0.0, appTask) |
```

**handleEntityDeleteCallback(entityID)**

```
|
+-----------------+------------+---------------------+
| mHC : HistoryController | appTask : AppearanceTask |
| handleEntityDeleteCallback(entityID, EInputType(0)) | create appearanceTask |
| create appTask(NULL, entityID, false, time (NULL)) |
| addTask(0.0, appTask) |
```
Figure 6.5.1 shows the sequences of our entityGenerator. It describes the flow of our program.

We will continue with the sequences describing the functionality of the join, simulate and resign function based on the above sequence.
We move on with the join sequence, this is the three of the last steps of the above sequence.

![Sequence join function diagram](Image)

Figure 6.5.2: Sequence join function

Moving on to the two of the last step, the simulation, this is where all the magic happens.

![Simulation sequence diagram](Image)

Figure 6.5.3: Part one of the simulation sequence.

Continue with sequence performTask see figure 6.5.4
Continue with appearance task sequence see figure 6.5.5

Continue with C2 task sequence see figure 6.5.8

Continue with addToAdministartionList sequence see figure 6.5.6

Continue with DeleteFromAdministartionList sequence see figure 6.5.7
Figure 6.5.6: addToAdministrationList sequence

Figure 6.5.7: DeleteFromAdministrationList sequence
Figure 6.5.8: C2 Task sequence

Continue with sequence handleC2DatabaseRequest see figure 6.5.9
Figure 6.5.9: handleC2DatabaseRequest sequence.
After performing the sequence in figure 6.5.4 till figure 6.5.9, the system continues with the sequence in figure 6.5.10.

![Simulation sequence diagram](image)

Figure 6.5.10: Repeat simulation loop

If the simulation loop ends then the system continues with last step of the flow, resign and close the system.

The last step of the sequence of figure 6.5.1 is resign.

![Sequence join function diagram](image)

Figure 6.5.11: Sequence join function
An important sequence is also to send the generated message to HLA.

Figure 6.5.12: sendAISToHla sequence.
Appendix F: MoSCoW Model

MoSCoW Model
## General

<table>
<thead>
<tr>
<th># Requirement</th>
<th>User/System</th>
<th>M</th>
<th>C</th>
<th>S</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ_GEN_001</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must be able to create area’s and assign one or multiple functions to it</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_003</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must be able to create routes and assign one or multiple functions to it</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_004</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must be able to generate entities for a route</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_005</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must be able to create spawn points and assign one or multiple functions to it</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_006</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must be able to generate entities for a spawn point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_007</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must be able to create points from where entities will be removed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_008</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must have the possibility to assign density to an area of route</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_009</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must store entity information in a data store</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_010</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user must be able to request information from a data store using queries</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_011</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user must be able to initialize a scenario based on a random start time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_012</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must have the possibility to import geographic areas, road and points</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_013</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must have the possibility to save the created scenario</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_014</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The scenario must contain an area of interest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GEN_015</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must be able to accelerate the simulation speed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GDP_001</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must simulate entity behavior, based on DMPs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GDP_002</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMPs must be described using finite state machine</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GNT_001</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must have an Entity generator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GNT_002</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The entity generator must have a varying output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GNT_003</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The DMPs must have a persistent behavior</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GRP_003</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user should be able to disconnect DMPs of entities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GSB_001</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user must have the possibility to execute REQ_GEN_001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GSB_002</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user must have the possibility to execute REQ_GEN_003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GSB_003</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user must have the possibility to execute REQ_GEN_005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GSB_004</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user must have the possibility to execute REQ_GEN_007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sea Traffic Generator

<table>
<thead>
<tr>
<th># Requirement</th>
<th>User/System</th>
<th>Interaction</th>
<th>M</th>
<th>C</th>
<th>S</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ_GSB_005</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user must have the possibility to execute REQ_GEN_012</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GSB_006</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user must have the possibility to execute REQ_GEN_013</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GSB_007</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user must have the possibility to execute REQ_GEN_015</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GRP_001</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user should be able to add entities manually</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_GRP_002</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The user should be able to change DMP of an entity</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Maritime Situation Awareness

<table>
<thead>
<tr>
<th># Requirement</th>
<th>User/System</th>
<th>Interaction</th>
<th>M</th>
<th>C</th>
<th>S</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ_MDP_001</td>
<td>S</td>
<td>VR Forces</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system must simulate vessel behavior, based on several DMPs</td>
<td>S</td>
<td>VR Forces &lt;&lt; Simobs logic ↔ DMPs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thesis Bachelor Project Sea Traffic Generator 175
<table>
<thead>
<tr>
<th>Request Code</th>
<th>Description</th>
<th>Operation</th>
<th>Query Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ_MDP_003</td>
<td>The system could reuse DMPs when operating in a different area.</td>
<td>S</td>
<td>SIMOBS logic ↔ DMPs</td>
</tr>
<tr>
<td>REQ_RCS_001</td>
<td>The user must be able to request Radar Cross Section information about an entity.</td>
<td>U</td>
<td>User ↔ using SOA ReST query ↔ RCS table</td>
</tr>
<tr>
<td>REQ_EOI_001</td>
<td>The user must be able to request EO / IR information about an entity from a table.</td>
<td>U</td>
<td>User ↔ using SOA ReST query ↔ EO/IR table</td>
</tr>
<tr>
<td>REQ_AIS_001</td>
<td>The user must be able to request AIS information for one or multiple entities.</td>
<td>U</td>
<td>User ↔ using SOA ReST query ↔ Entity Information DB</td>
</tr>
<tr>
<td>REQ_PIC_001</td>
<td>The user must be able to request wiki information about an entity.</td>
<td>U</td>
<td>User ↔ using SOA ReST query ↔ Wiki DB</td>
</tr>
<tr>
<td>REQ_INT_001</td>
<td>The user must be able to request INTELL report information by entity’s name or MMSI</td>
<td>U</td>
<td>User ↔ using SOA ReST query ↔ Entity Information DB</td>
</tr>
<tr>
<td>REQ_HMV_001</td>
<td>The user must be able to request the log of the entity-type modifications</td>
<td>U</td>
<td>User ↔ using SOA ReST query ↔ Entity Information DB</td>
</tr>
<tr>
<td>REQ_HMV_002</td>
<td>The system must be able to keep a log of the entity-type modifications</td>
<td>S</td>
<td>User ↔ using SOA ReST query ↔ Entity Information DB</td>
</tr>
<tr>
<td>REQ_HMV_003</td>
<td>The user must be able to request stored position(s) of the ships</td>
<td>U</td>
<td>User↔ using SOA ReST query ↔ Entity Information DB</td>
</tr>
<tr>
<td>REQ_HMV_004</td>
<td>The system must be able to store position(s)</td>
<td>S</td>
<td>User ↔ using SOA ReST query ↔ Entity Information DB</td>
</tr>
<tr>
<td>REQ_HMV_005</td>
<td>The system must be able to generate past position(s)</td>
<td>S</td>
<td>User ↔ using SOA ReST query ↔ Entity Information DB</td>
</tr>
<tr>
<td>REQ_HMV_006</td>
<td>The user must be able to request stored crew/ownership</td>
<td>U</td>
<td>User ↔ using SOA ReST query ↔ Entity Information DB</td>
</tr>
<tr>
<td>REQ_HMV_007</td>
<td>The system must be able to store crew/ownership</td>
<td>S</td>
<td>User ↔ using SOA ReST query ↔ Entity Information DB</td>
</tr>
<tr>
<td>REQ_MAR_001</td>
<td>The user should be able to request one or multiple Alpha report(s)</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>REQ_MAR_002</td>
<td>The system should be able to store Alpha report(s)</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>REQ_MAR_003</td>
<td>The system should be able to generate Alpha report(s)</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>REQ_MDP_002</td>
<td>The system should generate (new) history when the user modified the DMP of a vessel (using REQ_MRP_002).</td>
<td>S</td>
<td>VR Forces GUI ↔ SIMOBS logic ↔ DMPs ↔ Entity Information DB</td>
</tr>
<tr>
<td>REQ_MDP_004</td>
<td>Every vessel should have a unique behavior</td>
<td>S</td>
<td>VR Forces GUI ↔ SIMOBS logic ↔ DMPs</td>
</tr>
<tr>
<td>REQ_MRP_001</td>
<td>The actions of the role player should be stored.</td>
<td>S</td>
<td>VR Forces GUI (action) ↔ Scenario DB</td>
</tr>
<tr>
<td>REQ_MRP_002</td>
<td>The role player should be able to execute an INTELL report.</td>
<td>U</td>
<td>User ↔ using SOA ReST query ↔ INTELL Reports DB</td>
</tr>
<tr>
<td>REQ_MRP_003</td>
<td>The role player should be able to create Alpha report with required data</td>
<td>U</td>
<td>User ↔ using SOA ReST query ↔ INTELL Reports DB</td>
</tr>
<tr>
<td>REQ_MHI_001</td>
<td>The system should able to provide a complete log for the Sea Traffic Generator</td>
<td>S</td>
<td>AIS Generator + Entity Information DB + INTELL Reports DB</td>
</tr>
<tr>
<td>REQ_AIS_002</td>
<td>The system should be able to generate specific AIS information about an entity.</td>
<td>S</td>
<td>VR Forces ↔ SIMOBS logic ↔ DMPs ↔ SIMOBS logic ↔ AIS Generator ↔ Entity Information DB</td>
</tr>
</tbody>
</table>
Appendix G: AIS messages information

This appendix contains information about the AIS messages of our system. The next pages are cut from an original AIS document, which received from Eric den Breejen. Other parts of the document were not relevant for our system.
### 3.3.8.2.1 Messages 1, 2, 3: position reports

The position report should be output periodically by mobile stations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Number of bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message ID</td>
<td>6</td>
<td>Identifier for this message 1, 2 or 3</td>
</tr>
<tr>
<td>Repeat indicator</td>
<td>2</td>
<td>Used by the repeater to indicate how many times a message has been repeated. Refer to § 4.6.1; 0-3; 0 = default; 3 = do not repeat any more</td>
</tr>
<tr>
<td>User ID</td>
<td>30</td>
<td>MMSI number</td>
</tr>
<tr>
<td>Navigational status</td>
<td>4</td>
<td>0 = under way using engine, 1 = at anchor, 2 = not under command, 3 = restricted manoeuvrability, 4 = constrained by her draught, 5 = moored, 6 = aground, 7 = engaged in fishing, 8 = under way sailing, 9 = reserved for future amendment of navigational status for ships carrying DG, HS, or MP, or IMO hazard or pollutant category C (HSC), 10 = reserved for future amendment of navigational status for ships carrying DG, HS or MP, or IMO hazard or pollutant category A (WIG); 11-14 = reserved for future use, 15 = not defined = default</td>
</tr>
<tr>
<td>Rate of turn ROT\textsubscript{AIS}</td>
<td>8</td>
<td>$\pm 127 \sim 128$ (80\textdegree h) indicates not available, which should be the default. Coded by $\text{ROT}<em>{\text{AIS}} = 4.733 \sqrt{\text{ROT}</em>{\text{INDICATED}}}$ degrees/min $\text{ROT}_{\text{INDICATED}}$ is the rate of turn (720\textdegree/min), as indicated by an external sensor. $+127 = \text{turning right at 720}\textdegree/\text{min or higher}$ $-127 = \text{turning left at 720}\textdegree/\text{min or higher}$</td>
</tr>
<tr>
<td>SOG</td>
<td>10</td>
<td>Speed over ground in 1/10 knot steps (0-102.2 knots) $1023 \sim \text{not available}, 1022 = \text{1022 knots or higher}$</td>
</tr>
<tr>
<td>Position accuracy</td>
<td>1</td>
<td>$1 = \text{high (}&lt;10 \text{ m}; \text{differential mode of e.g. DGNSS receiver}) 0 = \text{low (}&gt;10 \text{ m}; \text{autonomous mode of e.g. global navigation satellite system (GNSS) receiver or of other electronic position fixing device}); 0 = \text{default}$</td>
</tr>
<tr>
<td>Longitude</td>
<td>28</td>
<td>Longitude in 1/10 000 min (±180\textdegree, East = positive, West = negative. 181\textdegree (6791AC0\textdegree h) = not available = default)</td>
</tr>
<tr>
<td>Latitude</td>
<td>27</td>
<td>Latitude in 1/10 000 min (±90\textdegree, North = positive, South = negative. 91\textdegree (3412140\textdegree h) = not available = default)</td>
</tr>
<tr>
<td>Message ID</td>
<td>Communication state</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SOTDMA communication state as described in § 3.3.7.2.2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SOTDMA communication state as described in § 3.3.7.2.2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ITDMA communication state as described in § 3.3.7.3.2</td>
<td></td>
</tr>
</tbody>
</table>

### TABLE 15b

<table>
<thead>
<tr>
<th>Message ID</th>
<th>Communication state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SOTDMA communication state as described in § 3.3.7.2.2</td>
</tr>
<tr>
<td>2</td>
<td>SOTDMA communication state as described in § 3.3.7.2.2</td>
</tr>
<tr>
<td>3</td>
<td>ITDMA communication state as described in § 3.3.7.3.2</td>
</tr>
</tbody>
</table>

### 3.3.8.2.3  Message 5: Ship static and voyage related data

Should only be used by Class A shipborne mobile equipment when reporting static or voyage related data.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Number of bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message ID</td>
<td>6</td>
<td>Identifier for this message 5</td>
</tr>
<tr>
<td>Repeat indicator</td>
<td>2</td>
<td>Used by the repeater to indicate how many times a message has been repeated. Refer to § 4.6.1; 0-3; 0 = default; 3 = do not repeat any more</td>
</tr>
<tr>
<td>User ID</td>
<td>30</td>
<td>MMSI number</td>
</tr>
<tr>
<td>AIS version indicator</td>
<td>2</td>
<td>0 = station compliant with AIS edition 0; 1-3 = station compliant with future AIS editions 1, 2, and 3</td>
</tr>
<tr>
<td>IMO number</td>
<td>30</td>
<td>1-999999999; 0 = not available = default</td>
</tr>
<tr>
<td>Call sign</td>
<td>42</td>
<td>7 × 6 bit ASCII characters, @@@@@@@ = not available = default</td>
</tr>
<tr>
<td>Name</td>
<td>120</td>
<td>Maximum 20 characters 6 bit ASCII, @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ = not available = default</td>
</tr>
<tr>
<td>Type of ship and cargo type</td>
<td>8</td>
<td>0 = not available or no ship = default 1-99 = as defined in § 3.3.8.2.3.2 100-199 = preserved, for regional use 200-255 = preserved, for future use</td>
</tr>
<tr>
<td>Dimension/ reference for position</td>
<td>30</td>
<td>Reference point for reported position. Also indicates the dimension of ship (m) (see Fig. 18 and § 3.3.8.2.3.3)</td>
</tr>
<tr>
<td>Type of electronic position fixing device</td>
<td>4</td>
<td>0 = undefined (default) 1 = GPS 2 = GLONASS 3 = combined GPS/GLONASS 4 = Loran-C 5 = Chayka 6 = integrated navigation system 7 = surveyed 8-15 = not used</td>
</tr>
<tr>
<td>ETA</td>
<td>20</td>
<td>Estimated time of arrival; MMDDHHMM UTC Bits 19-16: month; 1-12; 0 = not available = default Bits 15-11: day; 1-31; 0 = not available = default Bits 10-6: hour; 0-23; 24 = not available = default Bits 5-0: minute; 0-59; 60 = not available = default</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Maximum present static draught</td>
<td>8</td>
<td>in 1/10 m, 255 = draught 25.5 m or greater, 0 = not available = default; in accordance with IMO Resolution A.851</td>
</tr>
<tr>
<td>Destination</td>
<td>120</td>
<td>Maximum 20 characters using 6-bit ASCII; @@@@@@@@@@@@@@@@@@@@@@@@@ = not available</td>
</tr>
<tr>
<td>DTE</td>
<td>1</td>
<td>Data terminal ready (0 = available, 1 = not available = default)</td>
</tr>
<tr>
<td>Spare</td>
<td>1</td>
<td>Spare. Not used. Should be set to zero</td>
</tr>
<tr>
<td>Number of bits</td>
<td>424</td>
<td>Occupies 2 slots</td>
</tr>
</tbody>
</table>

This message should be transmitted immediately after any parameter value has been changed.
Appendix H: Sample of code

Sample of Code
Sample of code

In this section we explain some codes from the system, which are worth notifying. Such as determining the missing AIS data and how (different type of) task are coupled to the right generators. In this chapter we will give a closer look at particularly parts of the source code.

C2 request: C2Model::hasOverlap

This function is pretty interesting since we use it to determine whether a new appearance point falls in an already existing appearance interval. First we call a function to get appearance data from the database stored in a ResultSet. Based on this result set we determine it a point has overlap. The following possibilities can occur:

Explanation APPEARANCE TABLE: LEFT: 1 (IN), RIGHT: 0 (OUT)

- If there are no point on left and right, then there is no overlap
- If left has an 1, then this point is overlapping known data
- If right has an 0, then this point is overlapping known data

```cpp
bool C2Model::hasOverlap( sql::ResultSet * r, Eboundary bound )
{
    // als het punt met deze tijd niet bestaat
    if( r->rowsCount() == 0 )
    {
        SAFE_DELETE(r);
        return false;
    }
    else if( r->rowsCount() > 1 )
    {
        r->beforeFirst();
        while( r->next() )
        {
            // left boundary
            if (bound == 0 && r->getInt(2) == 0 )
            {
                std::cout << r->getInt(1) << std::endl;
                SAFE_DELETE(r);
                return false;
            }
            // right boundary add
            if (bound == 1 && r->getInt(2) == 1 )
            {
                std::cout << r->getInt(1) << std::endl;
                SAFE_DELETE(r);
                return false;
            }
        }
    }
    SAFE_DELETE(r);
    return true;
}
```

Figure 4.2: implementation C2Model::hasOverlap
C2Model::determineGapsFromResultSet
This function determines the gaps based on a appearance interval. This function is needed for generating missing AIS data when C2 request for it. The following possibilities could occur:

- Gap(s) before first appearance
  *For example C2 request for data beginning from point x to y. But the database only has appearance data from point x+1 to y. The missing data between x and x+1 needs to be filled.*

- Gap(s) between last appearance and end
  *For example C2 request for data beginning from point x to y. But the database only has appearance data from point x to y-1. The missing data between y-1 and y needs to be filled.*

- Gap(s) between appearances
  *For example C2 request for data beginning from point x to y. But the database only has appearance data from point x to x+1 and x+3 till y. The missing data between x+1 and x+3 needs to be filled.*
TupleVector * C2Model::calculateGapsUsingResultSet(sql::ResultSet * r, int begin, int end) {
    // initialize
    int temp = 0, i = 0;
    TupleVector * intervalVec = new std::vector<std::pair<int, int>>();

    // if the ships has at least once been appeared
    if (r->rowCount() > 0) {
        // determine the intervals in which we generated data for the
        // entities (IN, OUT)
        while (r->next()) {
            // first and 1
            if (i == 0 && r->getInt(2) == 1) {
                // gap(s) before first appearance
                intervalVec->push_back(std::pair<int, int>(begin, r->getInt(1)));
            }
            // first and 0
            else if (i == 0 && r->getInt(2) == 0) { /* do nothing */ }
            // last and 0
            else if (i == (r->rowCount()-1) && r->getInt(2) == 0) {
                // gap between last appearance and end
                intervalVec->push_back(std::pair<int, int>(r->getInt(1), end));
            }
            // other possibilities sequence
            else if (r->getInt(2) == 1) {
                // gap(s) between appearances
                intervalVec->push_back(std::pair<int, int>(temp, r->getInt(1)));
            }
            temp = r->getInt(1);
            i++;
        }
        // representation
        showGapRepresentation(intervalVec);
        // clean up
        SAFE_DELETE(r);
        return intervalVec;
    } else {
        SAFE_DELETE(r);
        intervalVec->push_back(std::pair<int, int>(begin, end));
        return intervalVec;
    }
    SAFE_DELETE(r);
    return NULL;
}

Figure 4.3: implementation C2Model::determineGapsFromResultSet
This function calculates interval lengths and determine timestamp of new tasks and add the tasks to tasklist.

Determine check for every interval:
- Width of the gap
- The number of tasks based on the width divided by the heartbeat of the message (determined by the generator), the number of recursions.
- Determine the starttime by placing the block with task in the middle of the gap and getting the right value of this block.
- Create a new task based on these values and add these tasks to the administration and tasklist.

```cpp
void C2Model::generateTasksBasedOnGaps( Generator * g, TupleVector * intervalVec,
  std::string entityID )
{
    double heartbeat = g->getHeartBeat();

    int gap = 0, c = 0;
    double taskTime = 0;
    TupleVector::iterator cit = intervalVec->begin();
    while (cit != intervalVec->end() )
    {
      // calculate length of the interval with missing information
      gap = (*cit).second - (*cit).first;

      // calculate the number of message-heartbeats fit in this interval
      (int) c = ceil( (double) (gap/heartbeat) );

      // calc latest time
      taskTime = (*cit).second - (( gap - (c*heartbeat) ) / 2);
      Task * c2Task = new C2RequestSubTask( g, entityID, taskTime, c );

      // add to registration
      mHc->addToC2SubTaskRegistration( c2Task );
      // TODO: distribution subtasks in certain given maximum time
      mHc->addTask( 0.0, c2Task );
      cit++;
    }
}
```

Figure 4.4: implementation C2Model::generateTasksBasedOnGaps()

This function:
- Determine the generators enable to generate for this inputtype (c2)
- Check whether the generators are able to generate for entitytype based on tableName (in tables)
  - For every entity and for every generator remaining generator:
    - Determine gaps in our data
    - Modify our administration
    - Create tasks in order to generate messages to fill the gaps
  - If no tasks has to be created in order to satisfy request:
    - Send back acknowledgement
void C2Model::handleC2DatabaseRequest( Interval * interval, DBTableSet * tables, EntitySet * entities )
{
    bool newBoundaryLeft, newBoundaryRight;
    sql::ResultSet * entAppearanceData;
    sql::ResultSet * leftBoundaryData;
    sql::ResultSet * rightBoundaryData;

    // set c2 parameters
    mInterval = interval;
    mTables = tables;
    mEntities = entities;

    // determine generator that are able to handle c2 callbacks
    std::vector<Generator*> * capableGenerators =
        mHc->getGeneratorModel()->getCapabilitiesForInputType(EinputType(1), tables);

    // determine for every entity combined with the able generator
    // the gaps in the requested data, add tasks
    // update the database after overlap check
    //
    // for every entity
    EntitySet::iterator eIt = entities->begin();
    while( eIt != entities->end() )
    {
        // ... and for every capable generator
        std::vector<Generator*>::iterator gIt = capableGenerators->begin();
        while( gIt != capableGenerators->end() )
        {
            newBoundaryLeft = false; newBoundaryRight = false;

            // obtain appearance data from database
            entAppearanceData =
                mHc->getDBHandler()->getAppearanceFromDBWithInterval(*eIt, interval->first, interval->second, (*gIt)->getTableName());

            // calculate gaps
            TupleVector * gaps =
                this->calculateGapsUsingResultSet( entAppearanceData, interval->first, interval->second );

            if( gaps->size() > 0 )
            {
                // clean up redundant old intervals that where located in
                //the new interval + boundary points
                ConditionSet * conSet = new ConditionSet();
                conSet->push_back( Condition( "=",
                    st::pair<std::string, std::string>("TIME_IN_SEC",
                        Convert::integerToString( interval->first ) ) ) );
                conSet->push_back( Condition( ",=
                    st::pair<std::string, std::string>("TIME_IN_SEC",
                        Convert::integerToString( interval->second ) ) ) );
                mHc->getDBHandler()->deleteFromDatabase( conSet,
                    mHc->getTableNameBasedOnInputType(*gIt)->getTableName(), EinputType(1) ), (*eIt) );

                // obtain boundary information at the left interval point
                leftBoundaryData =
                    mHc->getDBHandler()->getAppearanceBoundaryData( (*eIt), interval->first,
                        mHc->getTableNameBasedOnInputType(*gIt)->getTableName(), EinputType(1) ), Eboundary(0) );
            }
        }
    }
}
// determine whether a new begin/appearance point needs to be saved for the left boundary
if (this->hasOverlap(leftBoundaryData, Eboundary(0)) == false)
{
    mHc->getDBHandler()->storeEntityAppearance((*eIt), 1, interval->first, 
        mHc->getTableNameBasedOnInputType(*gIt->getTableName(), EinputType(1)), 
        newBoundaryLeft = true;
}

// obtain boundary information at the right interval point
rightBoundaryData = mHc->getDBHandler()->getAppearanceBoundaryData(*eIt, 
    interval->second, mHc->getTableNameBasedOnInputType(*gIt->getTableName(), EinputType(1)), Eboundary(1));

// determine whether a new end/disappearance point needs to be saved for the right boundary
if (this->hasOverlap(rightBoundaryData, Eboundary(1)) == false)
{
    mHc->getDBHandler()->storeEntityAppearance(*eIt, 0, interval->second, 
        mHc->getTableNameBasedOnInputType(*gIt->getTableName(), EinputType(1)), 
        newBoundaryRight = true;
}

// if new tasks has to be generated
if (newBoundaryLeft || newBoundaryRight)
{
    // create tasks in order to fill the gaps (missing data)
    this->generateTasksBasedOnGaps(*gIt, gaps, (*eIt));
}

gIt++;
}
eIt++;

// the requested data doesn't contain gaps
if (mHc->getC2SubTaskRegistration()->size() == 0)
    mHc->sendRequestedDataToC2();

Figure 4.5: implementation HistoryController::handleC2DatabaseRequest()
GeneratorModel

- Equal Distribution Of StartTime
  Every time a new task must be inserted it makes use of a start time determined by this function. Every consecutive task will be performed one step after the one before and depend on its heartbeat it will circular increment: 6, 7, 8, 9, 10, 0, 1, 2, 3

- Ensuring a proper start time per generator when a new task has not been created for longer than one second i.e. 5 tasks where generated 5 seconds ago, (0\rightarrow t_1, 1\rightarrow t_2, ..., 4\rightarrow t_5) now (5), a new task needs to be executed, so the new time needs to be 0 because all tasks has been performed in the foregoing seconds

```cpp
double GeneratorModel::determineStartTime( Generator * g )
{
    double startTime = g->getStartTime();
    // modify new startime next task
    double updateTime = startTime + 1.0;
    if ( updateTime >= g->getHeartBeat() )
        updateTime = 0;
    g->setStartTime( updateTime );
    std::cout << "Gen: " << g->toString() << " startime: " << startTime
              << std::endl;
    return startTime;
}
```

```cpp
void GeneratorModel::updateStartTime( EInputType iType, double deltaTime )
{
    std::vector< Generator* > * genList =
        getGeneratorListBasedOnInputType( iType );
    std::vector< Generator* >::iterator it = genList->begin();
    while (it != genList->end())
    {
        if ( (*it)->hasToExecute( iType ) )
        {
            (*it)->decreaseStartTime( deltaTime );
        }
        it++;
    }
}
```

Figure 4.6: implementation Equal Distribution Of StartTime

GeneratorModel::initEntityToGenerators
Every Generator determines whether it is able to handle a new task or not using the inputtype. After that it creates tasks using that Generator

```cpp
void GeneratorModel::initEntityToGenerators( std::string entityID, EInputType iType )
{
    //cout << " [GenModel] send task to all generators: \n" << endl;
    std::vector< Generator* >::iterator it = this->getGeneratorList()->begin();
    while (it != this->getGeneratorList()->end())
    {
        if ( (*it)->hasToExecute( iType ) )
        {
            Task * task = (*it)->createTask( entityID, iType );
            if ( task != 0 )
            {
                double time = this->determineStartTime( (*it) );
                mHc->addTask( time, task );
            }
        }
        it++;
    }
}
```

Figure 4.7: implementation GeneratorModel::initEntityToGenerators
Appendix I: MSA Block Diagram
Appendix K: SIG Evaluation

SIG Evaluation
(Dutch only)
First evaluation of SIG, by Eric Bouwers:
De code van het systeem scoort ruim drie sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De score wordt naar beneden gehaald door de Unit Size, de Duplicatie en het gebrek aan componenten.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de extreem lange methodes in dit systeem, zoals bijvoorbeeld de 'DBHandler::getAIS'-methode, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '// creating the statement from the given conditions' en '//'create default message' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Wat daarnaast opvalt is dat de langere methodes binnen dit systeem ook een hogere complexiteit-score hebben, wat lijkt voor te komen uit aparte functionaliteit voor het afhandelen van de twee verschillende bericht-types. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage reddantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem is er voornamelijk duplicatie te vinden tussen de source-files waarin de termen 'Message5' en 'Message123' voorkomen. In combinatie met de vorige observatie lijkt het erop dat er binnen het systeem nog een modelleerslag gedaan kan worden om duidelijker te maken welke functionaliteit uniek is voor elk bericht-type, en welke functionaliteit gedeeld kan worden tussen alle bericht-types. Het is aan te raden hier (nogmaals) over na te denken, of anders te documenteren waarom dit niet mogelijk is.

Wat daarnaast nog opvalt bij het bekijken van de code is dat er geen duidelijke componentenstructuur zichtbaar is op het file-systeem. Dit maakt het voor een ontwikkelaar in eerste instantie lastiger om een algemeen beeld te krijgen van de functionaliteit die het systeem aanbied. Wij raden aan om kritisch te overwegen om de code in verschillende (functionele) componenten op te delen om zo een eerste indruk te geven van de high-level structuur van het systeem.

Over het algemeen scoort de code gemiddeld, hopelijk lukt het om dit op niveau te blijven of dit te verbeteren tijdens de rest van de ontwikkel fase. De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Second evaluation of SIG:
In de tweede upload zien we dat de omvang van het systeem met 20 procent is toegenomen, daarbij is de score voor onderhoudbaarheid licht gestegen naar net vier sterren. Deze stijging is toe te schrijven aan een lichte stijging voor Unit Size en Unit Complexity. Daarnaast maakt de introductie van de componenten het makkelijker om de structuur van de applicatie te zien. Hierbij moet wel opgemerkt worden dat de reden voor alle afhankelijkheden tussen de componenten niet meteen duidelijk is, zie bijvoorbeeld de cyclische afhankelijkheid tussen de 'HistoryController' en de 'GeneratorModel'. Als laatste zien we dat het percentage duplicatie is toegenomen.

Uit deze observaties kunnen we concluderen dat het grootste deel van de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject. Het is goed om te zien dat naast een verbetering in de onderhoudbaarheid er ook nog steeds een stijging in het volume van de test-code te zien.
Appendix L: SQL versus NOSQL

SQL vs. NOSQL
DB Comparison: MySQL vs. NoSQL

De aanleiding van dit onderzoek is het probleem dat database architectuur continue kan veranderen. Het is nog onduidelijk hoe de database definitie eruit moet zien, want het kan zijn dat deze in een later stadium sterk kan wijzigen.

Vraag: Onderzoek het gebruik van NoSQL (Not Only SQL databases) voor de SeaTrafficGenerator (STG)

Eisen: Ondersteuning voor Windows en te gebruiken met C#/C++. Daarnaast moet de database eenvoudig onderhoudbaar zijn en een lage leercurve hebben voor het gebruik ervan. Liefst ook stabiel onder 32 bit systeem.

Probleemstelling: binnen MSA gebruiken we eenvoudige sensormodellen om te bepalen wat onze eenheden kunnen detecteren. Iedere sensor (camera, ESM, radar) vereist andere features/karakteristieken van een schip om te bepalen wat het kan detecteren, zoals aantal masten of bemanningsleden, radar cross sectie, wel of geen ladder aan boord, contouren, etc. Ook dient ieder schip een bepaalde variatie te bezitten t.o.v. de standaard karakteristieken, hetgeen ze unieker maakt, bv een bepaald schip heeft andere type zeilen, waarmee het zijaanzicht groter wordt. Verder zijn sommige karakteristieken afhankelijk van de afstand en oriëntatie, iets wat de STG niet weet, zodat bv een berekening van een radar cross section gedaan wordt met een afstand en oriëntatie afhankelijke formule. Tenslotte moet het mogelijk zijn dat de karakteristieken veranderen tijdens het scenario, zoals een schip dat de zeilen strikt.

Voorgestelde oplossing: Werken met twee databases (die liefst in hetzelfde database product draaien, bv beiden in MySQL of een NoSQL variant).

1. Een database bevat de standaard karakteristieken van een bepaald schip
2. Een database bevat de unieke en dynamische karakteristieken van een bepaald schip (bv strijken van de vlag)

Met iedere schip wordt een uniek ID meegestuurd. M.b.v. dit id kunnen de dynamische karakteristieken (in database 2) opgevraagd worden. Tevens definiëren wordt er een bericht gedefinieerd dat aangeeft welke ids gewijzigd zijn. Dus wanneer het schip met id=12345 zijn zeilen strikt, dan wordt er een bericht verstuurd dat id=12345 is veranderd. Daarnaast wordt er een bericht gedefinieerd dat aangeeft welke ids gewijzigd zijn. Dus wanneer het schip met id=12345 zijn zeilen strikt, dan wordt er een bericht verstuurd dat id=12345 is veranderd.

Vergelijking van de diverse databases:

<table>
<thead>
<tr>
<th></th>
<th>MySQL</th>
<th>NOSQL: MongoDB</th>
<th>NOSQL: Redis</th>
<th>NOSQL: couchDB</th>
<th>NOSQL : Cassandra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational database?</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Fixed schema</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>GUI admin tool</td>
<td>PhpMyAdmin</td>
<td>PhpMoAdmin</td>
<td>AdminUI</td>
<td>Kanapases IDE</td>
<td>Cassandra Cluster Admin</td>
</tr>
<tr>
<td>Leadership</td>
<td>Oracle</td>
<td>10Gen</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Possible solution:


<table>
<thead>
<tr>
<th>Scaling</th>
<th>Horizontal</th>
<th>Horizontal</th>
<th>Horizontal</th>
<th>Horizontal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name definitions</td>
<td>Table</td>
<td>Collection</td>
<td>Keys</td>
<td>Collection</td>
</tr>
<tr>
<td>Row or record</td>
<td>Document or object</td>
<td>Row keys</td>
<td>Document or object</td>
<td>Row keys</td>
</tr>
<tr>
<td>Column</td>
<td>Node or doc object</td>
<td>Node or doc object</td>
<td>column families</td>
<td></td>
</tr>
<tr>
<td>Plugin voor C++</td>
<td>C++ Connector</td>
<td>cxx-driver [not tested]</td>
<td>Redis + LibUV [prototype]</td>
<td>No direct native languages support drivers</td>
</tr>
<tr>
<td>Supports:</td>
<td>C++</td>
<td>C++ / C# [driver]</td>
<td>C++</td>
<td>Erlang</td>
</tr>
<tr>
<td>Known limitation(s)</td>
<td>Architecture of DB must be known. Win32 w/ FAT/FAT32 2GB/4GB Win32 w/ NTFS 2TB (possibly larger)</td>
<td>Total storage size for the server (data, indexes, everything) is 2.0gb [on 32 bits] If you are running on a 64-bit os, there is virtually no limit to storage size</td>
<td>Memory, not suitable to process a large amount of data. Total: 2 gb [on 32 bits]. For other system it depends on memory of the system. [RAM]. In most cases this is limited to 4 gb</td>
<td>Same as redis. You need to create views for each and every query, i.e. ad-hoc like queries (such as concatenating dynamic WHERE’s and SORT’s in an SQL) queries are not available it is possible to store binary data, but object sizes are limited by what can easily fit in ram. Which is mostly the same as Mongo,ouchDB.</td>
</tr>
<tr>
<td>Known Advantage</td>
<td>MySQL is characterised as a free, fast, reliable open source relational database and it can easily setup in Windows,Unix,Lunix. Using a tiny amount of memory. Advanced query operations, very quickly.</td>
<td>Advanced query operations, very quickly.</td>
<td>Your data will be indexed exactly for your queries, so you will get results in constant time</td>
<td>Ability to handle high insert workloads, high availability</td>
</tr>
<tr>
<td>Operation speed read/write</td>
<td>+/-</td>
<td>++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Datamapping (Speed)</td>
<td>+/-</td>
<td>++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Type of application</td>
<td>Serverapplications</td>
<td>Serverapplications</td>
<td>Serverapplications / tasks</td>
<td>Serverapplication</td>
</tr>
<tr>
<td>Possibility to replicate</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Losing data probability</td>
<td>+/-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>How quickly compatible with our program</td>
<td>++</td>
<td>-</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Database Type</td>
<td>Document store</td>
<td>Discrete key</td>
<td>Document store</td>
<td>Discrete key</td>
</tr>
<tr>
<td>Learning curve</td>
<td>++</td>
<td>+/-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Advices from experts</td>
<td>Is still used by lots of programs. If you know</td>
<td>Use if you have a ton of single types of documents</td>
<td>Use if you need a ton of counters</td>
<td>For accumulating, occasionally changing data, on</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If you really want to scale and your data schema allows for</td>
</tr>
</tbody>
</table>
De belangrijkste eis is de limiet van de database. Met de onderstaande berekening is te zien dat limiet van de database van groot belang is voor ons.

Stel er zijn 2000 schepen die allemaal ais berichten moeten bevatten van de afgelopen 30 dagen. Dan geldt er:

- Een bepaald type(message1,2,3) ais bericht moet een keer in de 2 a 10 seconde opgeslagen worden, en dit moet per schip gedaan worden.

  Uitgaande van 2 seconde:

  Per dag moet er dus 30*60*24 = 43200 berichten opgeslagen worden.

  Per schip moet dus 43200*30=1296000 berichten (voor 30 dagen records) opgeslagen worden in de database.

  Voor 2000 schepen geldt dus dat er 2592000000 berichten moeten worden. En dit alleen al voor 1 type AISmessage. In de MYSQL database komt 100000 records overeen met ongeveer 10 MB aan data. Met een kleine berekening is te zien dat 2592000000 records ongeveer overeenkomt met 259200MB aan data.

Er moet ook nog data opgeslagen worden van andere AIS message types, wat ook groot opslagruimte nodig heeft. Kortom het is dus van belang dat er genoeg ruimte beschikbaar is om data op te slaan.

Conclusie:

Weegt het ongemak van MySQL wel op tegen de zware nadelen die de NOSQL-systemen hebben, zoals zware beperkingen van de grootte van de totale database en het ongemakkelijk gebruik maken van queries.

MongoDB hebben wij werkend, inclusief de admin gui. Wel alleen als los programma maar nog niet in ons project geïntegreerd. Wij krijgen Redis niet werkend, die niet officieel support biedt voor windows. Hetzelfde geld ook voor CouchDB, waarbij er vele programma’s geïnstalleerd moeten worden. En niet elke programma succesvol wilt installen/compilén. ( door de ingewikkelde samenhang tussen de 7 programma’s en niet alle programma’s zijn compatible met een 32/64bit systeem.). Cassandra zou een goede keus geweest zijn als de lib compatible is en goed ondersteund wordt in Windows, echter is dit niet het geval.Gezien op basis van de initiële requirements MySQl een prima keuze is. Raden wij raden het NIET aan om NOSQL te gebruiken, om de volgende redenen: Aan MongoDB zit datalimit, namelijk 2GB ( bij gebruik van een 32 bit systeem), couchDB heeft een limiet van 2TB. Dit is voor onze opslag van data onvoldoende ook omdat wij werken op een 32 bit systeem. Een belangrijke beperking voor de meeste nosql databases is “the notion of consistency is sacrificed for

<table>
<thead>
<tr>
<th>Windows support</th>
<th>Yes</th>
<th>Yes</th>
<th>There is no official support for Windows builds</th>
<th>Yes</th>
<th>YES, but lib is not tested in windows, so might not be compatible.</th>
<th>it, Cassandra is probably the best solution</th>
</tr>
</thead>
</table>
Thesis Bachelor Project Sea Traffic Generator

198

performance and scaling. Data can diverge amongst it is copies and be inconsistent from time to time.” Geen van de onderzochte databases voldoen aan de gestelde eisen. Bovendien denken wij dat het probleem beschreven in het mailtje (wellicht) verholpen kan worden door nieuwe kolommen in de SQL database aan te maken, waarin de waarde van de nieuwe “features” opgeslagen kunnen worden. En bij het gebruik van een NOSQL database de gebruiker zelf de database “goed” beheren en onderhouden om een duidelijk overzicht te hebben. Wat er ook nog bijkomt, is het verlies van data, waardoor replicaties gemaakt moeten worden, dit vergt ook aardig veel tijd en bovendien kunnen replicaties van grote databases falen. Daarnaast denken wij dat het ook niet verstandig is om de NOSQL database te gebruiken, gezien de hoeveelheid tijd die nodig is om het werkkend te krijgen met wat wij nu hebben. Met andere woorden er geen tijd meer is om over te stappen.

Maar indien er NOSQL DB gebruikt “wilt” gaan worden in de toekomst dan raden wij Mongo aan, omdat dit gebaseerd op de eigenschappen het meest stabiel is van de drie en de leercurve laag is zoals gesteld in de opdracht en in een 64 bits systeem er geen data beperking is. En dat de database goed (handmatig)onderhouden wordt is natuurlijk ook een pre. Maar gezien de geringe tijd is deze optie nu helaas niet meer realiseerbaar.

References:
http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis
http://www.thoughtworks.com/articles/nosql-comparison
http://www.techrepublic.com/blog/10things/10-things-you-should-know-about-nosql-databases/1772
http://www.linuxjournal.com/article/10770
http://www.cloudcomputingetc.com/2012/05/i-say-no-to-nosql-big-data-solutions.html
MySQL:
Mongo:
http://blog.mongodb.org/post/137788967/32-bit-limitations
Redis:
https://gist.github.com/1439660
CouchDB:
http://wiki.apache.org/couchdb/installing_on_Windows
https://github.com/apache/couchdb/blob/master/INSTALL_Windows
http://www3.ntu.edu.sg/home/ehchua/programming/howto/cygwin_Howto.html
https://issues.apache.org/jira/browse/COUCHDB-690
http://www.nosql.se/tags/couchdb/
Cassandra:
http://scale.metaoptimize.com/questions/29/when-should-i-use-cassandra
http://snapwebsites.org/project/libqtcassandra
http://www.datastax.com/download/clientdrivers