

Eindverslag van de ontwerpersopleiding
Technische Informatica

Final report of the postgraduate programme
Software Technology

**Design and Implementation
of the
Mine Warfare Testbed
Geographical Information
System**

Drs. F.J.M. van Lingen

Supervisors: Ir. Paul de Krom MTD
TNO-FEL, The Hague

Dr. Ir. Huub van de Wetering
Eindhoven University of Technology

Augustus 1999
CIP-data, Koninklijke Bibliotheek, Den Haag

F.J.M. van Lingen

'Design and Implementation of the Mine Warfare Testbed Geographical Information System'

/Van Lingen, F.J.M.;

- Eindhoven:Stan Ackermans Instituut. -Ill.

Final report of the postgraduate programme Software Technology.

-With ref.

ISBN 90-5282-953-5

Subject headings: Geographical Information System/ Object Oriented Database

Copyright © August 1999, Stan Ackermans Instituut and TNO-FEL,

All rights reserved.

Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt door middel van druk, fotokopie, microfilm of op welke andere wijze dan ook zonder voorafgaande schriftelijke toestemming van de rechtshouder.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission from the copyright owner.

Abstract

The Mine Warfare Testbed (MWT) is a research environment currently being developed at TNO Physics and Electronics Laboratory (TNO-FEL), in order to enhance its mine warfare research efforts. The testbed provides a hardware and software environment for the evaluation and validation of systems, concepts and

tactics in the field of mine warfare. It defines standards to promote integration and re-use of research means and knowledge. As such, the MWT is intended to enhance the effectiveness and efficiency of future mine warfare research; this will allow TNO-FEL to react more quickly to changing customer demands. In addition, the testbed is to become a repository for future research results.

MWT-GIS is a research project, initiated to extend the MWT with GIS functionality. The *Objectstore* database tools were used to design an Object Oriented Spatial Database. Resulting in a design of a database that can contain arbitrary spatial objects. The graphical user interface displays the geographical environment stored in the database and the simulation entities of models of the MWT. These entities have to be displayed on the correct geographical position. Also a database management module was designed for managing the environments stored in the database. Furthermore, a module was designed to communicate with the MWT.

Keywords: *Geographical Information System, Simulation, Object Oriented Spatial Database, Mine Warfare Testbed.*

1. INTRODUCTION	7
1.1 TNO-FEL AND MINE WARFARE RESEARCH	7
1.2 MINE WARFARE TESTBED	7
1.3 REPORT OVERVIEW	8
1.4 ACKNOWLEDGEMENTS	8
2. PROJECT OVERVIEW	9
2.1 PROJECT HISTORY	9
2.2 PROJECT AIM	9
2.3 PROJECT MANAGEMENT	9
2.4 PROJECT PLANNING	9
3. ORIENTATION	11
3.1 OBJECTIVE.....	11
3.2 METHOD.....	11
3.2.1 Approach	11
3.2.2 Problems.....	11
3.2.3 Deliverables.....	11
3.3 RESULTS.....	11
3.3.1 MWT.....	11
3.3.2 User requirements	12
3.3.3 Feasibility.....	13
3.4 CONCLUSIONS	13
4. SYSTEM REQUIREMENTS.....	14
4.1 OBJECTIVE.....	14
4.2 METHOD.....	14
4.2.1 Approach	14
4.2.2 Problems.....	14
4.2.3 Deliverables.....	15
4.3 RESULTS.....	15
4.3.1 System concept.....	15
4.3.2 Database.....	16
4.3.3 Software requirements.....	21
4.4 CONCLUSIONS	22
5. DESIGN.....	23
5.1 OBJECTIVE.....	23
5.2 METHOD.....	23
5.2.1 Approach	23
5.2.2 Problems.....	23
5.2.3 Delivirables	23
5.3 RESULTS.....	24
5.3.1 Architectural design	24
5.3.2 Detailed design.....	25
5.3.3 Design deciscions	31
5.4 CONCLUSIONS	33
6. REALISATION	34
6.1 OBJECTIVE.....	34

6.2 METHOD.....	34
6.2.1 Approach.....	34
6.2.2 Problems.....	34
6.2.3 Deliverables.....	35
6.3 RESULTS.....	35
6.3.1 Database server.....	35
6.3.2 Simulation server.....	35
6.4 CONCLUSIONS	36
7. CONCLUSION.....	37
7.1 EVALUATION	37
7.1.1 Process.....	37
7.1.2 Overall.....	37
7.2 FUTURE WORK.....	38
GLOSSARY	40
LIST OF FIGURES.....	41
BIBLIOGRAPHY	42

1. Introduction

This final report describes a design project performed at TNO-FEL concerning the development of a geographical information system for the Mine Warfare Testbed (MWT). This chapter comprises a short introduction to TNO-FEL, a description of the MWT, and an outline of the rest of the report.

1.1 TNO-FEL and Mine Warfare Research

The TNO Physics and Electronics Laboratory (TNO-FEL) in The Hague is one of three institutes within TNO conducting defense research. This research is conducted mainly for the Dutch Ministry of Defense, but studies are also performed for other ministries, institutes, and companies. Currently, TNO-FEL has approximately 500 employees.

One of the clients of TNO-FEL is the Mine Warfare Service of the Royal Dutch Navy. The Mine Warfare Service is charged with carrying out Mine Countermeasures (MCM), i.e. mine hunting and mine sweeping operations, in order to reduce the risk for sea traffic caused by mines.

[Sorry picture got lost!]

Figure 1 The Maritime Operations Research pyramid of models and systems.

The Mine Warfare (MW) section of TNO-FEL is dedicated to carrying out research for the Mine Warfare Service. The hierarchical nature of MW research is reflected in the Maritime Operations Research pyramid depicted in Figure 1. The pyramid shows that Mine Warfare is supported at various levels. This is also the case for the other two areas of maritime warfare: Above Water Warfare (AWW) and Anti-Submarine Warfare (ASW). The acronyms of the models and systems which have been developed at TNO-FEL are displayed at each level. The hierarchical nature of the pyramid is such that lower-level models and systems provide the input for those at higher levels.

The contribution of the MW section lies in collecting information from various sources and models, and integrating that information in order to advise its clients. The studies that are carried out range from tactical studies concerning the deployment of multiple platforms (for example, the deployment of a mine hunter in a mine hunting operation) to system studies concerning the effectiveness and usage of single weapon systems (for example, the effectiveness of a new type of sonar). Summarizing, the MW research is primarily concerned with:

- The development of strategies and tactics concerning the deployment of Mine Countermeasures (MCM) units during operations.
- The evaluation and validation of both existing and newly developed MCM systems.
- The development of models relevant to mine warfare, such as models for underwater acoustics and mine burial.

1.2 Mine Warfare Testbed

The Mine Warfare Testbed (MWT) is a research environment currently being developed at TNO-FEL in order to enhance the MW research efforts. It is designed to facilitate the re-use and integration of systems, models, and applications in new research projects. The models that will be used on the testbed range from mission-analysis models to detailed physical models.

The MWT provides a hardware and software environment for the evaluation and validation of systems, concepts, and tactics in the field of MW. It defines standards to promote integration and re-use of research means and knowledge. As such, the MWT is intended to enhance the effectiveness and efficiency of future mine warfare research; this will allow TNO-FEL to react more quickly to changing customer demands. In addition, the testbed is to become a repository for future research results.

Currently, the MWT consists of the following parts:

- A C++ object framework for programming object-oriented entities (such as ships and mines), with coding and documentation standards.
- A simulation kernel, which provides simulation facilities such as reading configuration files and the communication between entities via events.
- The Simulation Modeling Environment (MWT-SME), which supports model developers and model users in building object-oriented MW models and performing simulations with these models.
- A model repository, in which all models built with the MWT-SME are stored.
- Three links have been developed: one link to another simulation environment (Electronic Battlefield Facility), and two links to existing models (MCM-RiskPlan and Mine Burial).

1.3 Report Overview

- Chapter 2 defines the design project, describing its goal, planning, and management.
- Chapter 3 describes the orientation phase in which the MWT was evaluated and user requirements were identified.
- Chapter 4 describes the system requirements and how they were obtained.
- Chapter 5 deals with the design phase, presenting the global design and discussing the design decisions that were made.
- Chapter 6 describes the implementation of the geographical information system for the MWT-SME. Chapter 7 contains the conclusions of the project.

1.4 Acknowledgements

First, I would like to thank TNO-FEL for providing me with the opportunity to carry out this final project. I would also like to thank my coaches Huub van de Wetering and Paul de Krom for their support and patience throughout the project. Although not actively involved in my project but certainly important, I would like to thank “groep 1-2” and especially “de mijnenbestrijders”. I would also like to thank Frank Benders for his support throughout the project. Special thanks goes out to Robert Moro and Jeroen Soede for helping me out. Last but not least, I would like to thank Corry, Marloes, and Harold.

2. Project Overview

This chapter gives an overview of the MWT-GIS design project. Firstly an overview of the history is given. Secondly, the aim of the design project is described. Thirdly, the project management is described. Finally, the phases in the planning of the project and their relations are shown.

2.1 Project History

The concept of the Mine Warfare Testbed was considered for the first time at the start of 1994. In December 1994, the design of the simulation kernel [Smak95+] was started by ir. C.H.J. Smakman MTD and ir. W.E.P. van der Sterren MTD. They specified the requirements of the MWT and developed the initial prototype of the simulation kernel. They concluded their work in September 1995.

The MWT Simulation Modeling Environment (MWT-SME; see [Bend96+]) project was the continuation of the MWT project and focused more on the user interaction with the MWT. This project was carried out by ir. F.P.A. Benders MTD and ir. P.P.J. de Krom MTD. They designed an interface to provide facilities to the user for developing and re-using models, for specifying simulations on the testbed, for execution of the simulations, and for the storage of all resources and results.

The MWT Linking project [Veld97] focused on the interaction of the MWT to external models and applications. In this project, links to three external systems had to be designed:

- A link to the Electronic Battlefield Facility (EBF), another simulation environment.
- A link to MCM-RiskPlan, a model for assisting in planning a Mine Countermeasures (MCM) operation in a situation where shipping is about to transit through a channel within limited time.
- A link to Mine Burial, a model for predicting the percentage of mine burial caused by liquefaction.

These links will serve as a model for future MWT links to resp. other simulation environments, MS Windows Dynamic Link Libraries (DLLs), and MatLab models. The MWT Linking project was carried out by ir. E.R. van Veldhoven MTD.

2.2 Project aim

The first three projects focused on an infrastructure for building simulation models, communicating with other models and executing these simulation models. Until this time environment variables were kept constant in the models. However, to make more realistic models, environment variables should be able to change with respect to the position and time. For example, depth of the North sea depends on the position and time (high tide and low tide). The MWT-GIS project integrates geographical data into the simulation environment. There are two levels of integration:

- Level 1: Creating a database for geographical data. This data can be used in simulations.
- Level 2: Visualize geographical data and simulation entities. Visualization leads to better analysis of the simulation models and the results they produce. To achieve level 2, level 1 is needed.

2.3 Project Management

The project was carried out by drs. F.J.M. van Lingen, student of the post-masters Software Technology Programme (Dutch abbreviation: OOTI) at the Eindhoven University of Technology. The project was led by ir. P.P.J. de Krom MTD at TNO-FEL. Supervision at the Eindhoven University of Technology was performed by dr. ir. H.M.M. van de Wetering. Support was provided by numerous TNO-FEL researchers active both in the domain of mine warfare and that of geographical information systems. They contributed during the analysis, requirements, and design phases, and aided in the implementation of the MWT GIS. During the project the MWT-GIS team was extended with ing. R. Moro. he assisted in the design and implementation phase.

2.4 Project Planning

In the planning of the MWT-GIS project, the following six phases have been distinguished:

- *Orientation phase*: this phase consisted of writing the project plan and familiarizing with the domain of mine warfare and the MWT. The user requirements were identified and a user requirements document was written
- *System Requirements phase*: in this phase, the system requirements were defined and the system requirements document was written.
- *Design phase*: in this phase, the design of the MWT-GIS was made and the design document was written.
- *Preparation phase*: this phase was intended to familiarize with the tools and languages to be used during the realization phase. Furthermore, a test document was written.
- *Realization phase*: this phase consisted of the implementation of the MWT GIS, writing the final report.
- *Test/Acceptation phase*: This phase consisted of testing the implementation, using the test document and giving presentations at TNO-FEL and the TUE.

Figure 2.1, is a graphical representation of the phases and the relation between different phases.

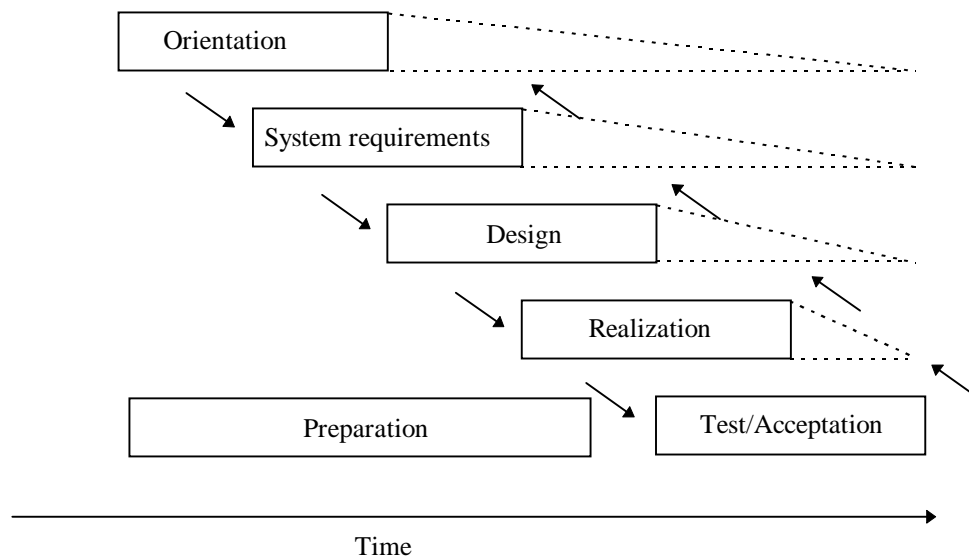


Figure 2. Phases in relation with time.

All phases were allowed to overlap in order to obtain early feedback on results of a specific phase. The dotted triangles at the end of several phases represent the fading out of these phases. Not every phase ends abruptly but becomes less demanding during the project. At the end of each phase, the deliverables of that phase were reviewed by the supervisors.

3. Orientation

The first phase of the design project was dedicated mainly to familiarize ourselves with the MWT-SME and the domain of mine warfare in general. Furthermore, a user requirements document was written.

3.1 Objective

The purpose of the orientation phase was to become familiar with the MWT-SME and mine warfare in general, and to refine the ideas concerning the MWT-GIS. Using this information and the information of the future users of the MWT-GIS a user requirements document was written. Furthermore, a project plan was made.

3.2 Method

3.2.1 Approach

At the beginning of the project we started reading all documentation about the MWT and the MWT-SME to gain insight into of the capabilities of the simulation kernel and the simulation modeling environment. In addition, we familiarized ourselves with the mine warfare domain at TNO-FEL. Furthermore, we talked with future users about their needs with respect to the MWT-GIS. This resulted into a user requirements document.

3.2.2 Problems

Two significant problems occurred during the orientation phase. The first problem was that it was difficult to get full insight in the capabilities of the MWT, since only one simulation model had been implemented, and the MWT-SME was still under development. The reports of the simulation kernel and the modeling environment only briefly cover simulation aspects, like the structure of an entity, the use of states and events, and the use of configuration files. However, the on-line documentation of the MWT was very useful in gaining knowledge about building simulations on the MWT kernel. As an additional problem, it took several weeks to get acquainted with the C++ compiler, linker, and Makefiles.

The second problem concerned the assignment. It was difficult to get a clear view of the needs of future users. Most of them had some ideas, but did not always know what to expect from the MWT-GIS. Furthermore, the MWDC would supply information for the MWT-GIS. But it was not clear what information that would be. During numerous discussions with future users it became more clear what they expected from the MWT-GIS. It turned out there were high expectations, and that the database that would be used (*Objectstore*) was not suited to fulfill all these expectations.

3.2.3 Deliverables

The effort put into the orientation phase resulted in:

- An entity in the MWT.
- A project plan
- A user requirements document [Ling1].

3.3 Results

This section gives a short description of the MWT. Furthermore, it summarizes the important aspects of the user requirement document and explains how feasible these aspects are.

3.3.1 MWT

The MWT architecture is displayed graphically in Figure 3. The MWT simulation kernel provides a discrete -event sequential simulation mechanism adopting an entity-oriented worldview. A simulation on the testbed is based on a (dynamic) model of the real world which can be composed of any number of

entities. These entities have their own state and can communicate with one another. Entities also have a concept of time: the state of an entity can change as time progresses, even when it does not communicate with other entities. In general, an entity will have a real world counterpart, such as a ship or a mine.

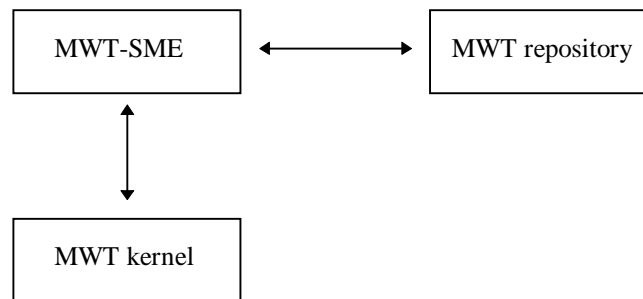


Figure 3 Overview of the MWT architecture

The MWT-SME is built on top of the MWT kernel and uses the facilities of the kernel to execute the simulations. It provides high-level facilities that aid the user in performing specific tasks, such as building models, defining scenarios, executing simulations, and analyzing simulations results.

The MWT repository stores all models, scenarios, and documentation, that are available to users. The information stored in the repository is re-used to compose new simulations in the MWT-SME. More details about the architecture of the MWT can be found in [Bend96+].

3.3.2 User requirements

The natural environment plays an important role in warfare operations. The most important environmental variables are: water temperature, salinity, pH value, depth, soil type and current. Besides environmental data also contact data is important. The values of these parameters do not only change when the position changes, but also change in time. Because of the complexity of these data, it is either not incorporated in the simulation or only used as a fixed value in time and place. It is important that during a simulation run this data can be used to get more realistic simulations. Besides simulations, route survey analysis, planning tools and burial analysis also need this environmental information for better analysis and more realistic planning.

Route survey analyzes contacts found during a route survey and compares it with contacts already found. Thus building a database containing known contacts. This information can be used for executing mine warfare operations in less time. Beside analyzing contacts, also the route itself is analyzed.

Planning tools make a planning of mine warfare operations. This plan includes several ships and ship types. Because not every ship can perform optimal in every area, environmental parameters are needed to identify the operation areas and the performance for every ship type

Burial analysis, analyzes the process of mine burial. To do this, information is needed of the soil type, porosity, wave height, wave heading.

To use the environmental information a database is needed. Also a query language has to be specified. All data is related to a time and a position. Besides a database, a visualisation tool is needed that can visualise the geographic data in combination with the entities of the simulation. A study was done to investigate if it was possible to link the MWT to a commercial GIS. It turned out that this was hard to do and it was decided to build a GIS from which MWT simulation could retrieve data and visualize it in an efficient manner. One of the problems was, that the access time to a regular geographic database was high. *Objectstore* and *Ilog views* are used for building the MWT-GIS. *Objectstore* is used to build the spatial database. *Ilog views* is used to build the user interface.

On the other hand there were requirements that were hard to satisfy using Objectstore but could be satisfied using a commercial GIS. This led to the conclusion that requirements had to be split into two groups:

- MWT-GIS requirements. Requirements with respect to incorporating environmental information in simulations.
- GIS requirements. Requirements that are related to commercial GIS packages. These requirements are more related to route survey, planning and burial analysis.

Requirements with respect to the MWT-GIS are described in [Ling1]

3.3.3 Feasibility

There are several problems in developing a GIS conform the requirements described above:

- Data size. Several parameters use large data sets. For example depth, current, and soil type. A database should be able to handle large geographical data sets efficiently.
- Query language. To do an analysis, data must be retrieved using a query language and conditions that contain relations between different parameters.
- Spatial analysis. It is impossible to store data of a parameter of every position and date. Therefore, spatial analysis needed to generate a one or two dimensional region.
- Performance. In a simulation it should not take a lot of time to retrieve information from a database, because it would decrease the performance of the simulation.

The database that was at our disposal was the *Objectstore* persistent object manager, and the spatial object manager extension. *Objectstore* offers the possibility to store object models. Furthermore, *Objectstore* is used in building the MWT and MWT-SME. Therefore, a GIS using *Objectstore* is easy to integrate with the MWT-SME environment. A drawback of *Objectstore* is the lack of a query language and the lack of support for a query language. Furthermore, you need spatial analysis tools for analyzing spatial data. *Objectstore* offered only spatial data structures.

3.4 Conclusions

During the orientation phase we obtained a lot of domain knowledge about mine warfare operations. We also obtained experience with the MWT-SME. We learned how to build entities and observers. Support was obtained from the designers of the MWT-SME.

The main result of the orientation phase was the MWT-GIS user requirements document [Ling1] and a study to general GIS requirements in the Mine Warfare section. It showed that the problem was considerable more complex than expected. It also led to the conclusion that *Objectstore* was not the database in which a GIS should be build. But that it could very well be used as a database to store environmental information during a simulation (MWT-GIS).

4. System Requirements

The second phase of the design project was dedicated to obtaining the software requirements for the MWT-GIS. In these requirements, the capabilities of the MWT-GIS are described in more detail. This chapter describes how the system requirements were determined, which problems were encountered, and which results were obtained.

4.1 Objective

The goal of the requirements phase was to produce a set of system requirements, based on an analysis of the user requirements of the MWT-GIS, GIS packages, the MWT and literature research. These requirements needed to be as complete, consistent, and correct as possible. In the System Requirements Specification [Ling2] the system requirements of the MWT-GIS are described.

4.2 Method

4.2.1 Approach

The User Requirements [Ling1] were used as a starting point for the system requirements. Using the User Requirements several system requirements and system concepts could be formulated. Besides this document, other sources of information were used.

To get an idea of a GIS, several GIS systems were examined. The first system was ArcView which is a commercial system. It is used by the MWDC in a pilot project that tries to determine requirements and gain knowledge of geographical information systems. The second system was the bathymetric system of the Hydrographic service. This is a specialized GIS for processing and analyzing data. The last system was a system developed at TNO-FEL using Illustra. This is an extended relational database, capable of handling geographical information. Besides examining geographical information systems the following information sources were used:

- The OpenGIS Simple Features Specification For SQL [Cons97]. This document describes how to incorporate and integrate GIS features with SQL databases. It also contained some object models that can be used in a more object oriented database environment.
- Library search. On the subject of GIS there was much information to be found. However, this information was mostly related to geography instead of computer science. Information about linking a GIS to a simulation environment was scarce.
- Discussions with future users. In order to find out what their ideas were. To get feedback from the future users user requirements were discussed, a prototype was build and several discussions were held.

4.2.2 Problems

Several problems were encountered when making the system requirements.

- Linking a GIS to a simulation environment is not done often. Therefore it was difficult to find information regarding this subject.
- Future users find it hard to describe requirements for the MWT-GIS because the models do not support it at this moment. During numerous discussions it also became clear that there was a need for a more 'standard' GIS that could visualize data for analysis rather than using it in simulations.
- Because users could not clearly describe what they want, flexibility was a requirement.
- Coordinate system. Because the earth is not flat and a map is, a projection is needed to display the earth on a map. Thus projecting coordinates of a sphere on a two dimensional Euclidian space. Every projection has its advantages and disadvantages with respect to the region it is used. Projections can be for instance area invariant or distance invariant. There are numerous projections and it takes a lot of time to find out which one can be used. It was decided to use UTM coordinates because it can be

represented as a pair of integers and computations with it are easy. Furthermore, accuracy of UTM coordinates on small areas is good. The area in which the simulation takes place is in most cases small.

4.2.3 Deliverables

The final result of the requirements phase was the system requirements document [ling2], containing the following:

- Global concepts regarding the structure of the object oriented database that will store geographic information.
- A short description of what the user interface will contain.
- A description of the system requirements for the complete application.

4.3 Results

This section first describes the database structure. Several concepts will be introduced to explain this. Secondly a description of the user interface/management will be given. Third the connection between the user interface, the database and the MWT-SME will be described. Finally a list of the main system requirements will be given.

4.3.1 System concept

One of the user requirements was that simulations of the MWT could use geographical information in their simulations. Furthermore, entities of a simulation can be made visible in the environment the simulation takes place. Another requirement was that a user of the MWT could define a scenario by using an environment. The scenario can be defined by graphically placing entities in the environment. The scenario's and models are stored in the MWT repository. These can be accessed directly by the MWT-GIS. However, the MWT already has an infrastructure for retrieving, storing and editing these models (MWT-SME). Thus linking the MWT-GIS to the MWT-SME results in this functionality without building a new infrastructure. Figure 4 shows the relation between the different MWT modules. The arrows indicate a data flow to another module. The MWT kernel represents the simulations interacting with the MWT-GIS module.

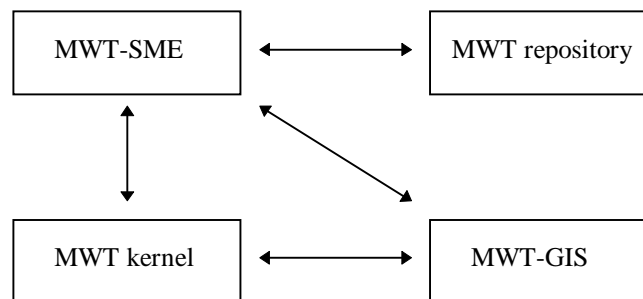


Figure 4. Relations between the different modules

In the MWT-GIS a user has the following options:

- **Simulation management.** A simulation can be started and managed. To achieve this, a scenario has to be loaded from the MWT repository via the MWT-SME. Furthermore, a user specifies the environment that will be used.
- **Scenario management.** A scenario can be defined by using a model stored in the MWT repository. The model is loaded by using the MWT-SME. Furthermore, an environment is selected from the database. This enables the user to graphically define a scenario by using a model and an environment.
- **Viewer.** If a simulation is running and an environment is selected a user can execute a viewer to visualise the simulation and the environment. Because an environment can be composed of many data layers that can not be made visible at the same time, it is possible to execute several viewers on

different machines. These viewers receive entity information from the simulation and will refresh the environment if necessary. For example, if a ship sails of the screen.

- **Environment management.** The environments in the database have to be managed. Thus a user can delete, modify, or insert a new environment. This new data can come from another geographical information system. This system contains several measurements of an area that are converted to a map that covers the whole area. After that, it is stored in the MWT spatial database.
- **Simulation server.** A server is needed to send the entity information of the simulation to the different viewers. Furthermore, this server enables users to select the functionality they want to use.

Figure 5 shows the different modules of the MWT-GIS and the relation with other MWT modules. The modules between the dotted lines are MWT-GIS modules. The arrows indicate a data flow or communication to a module. Some arrows are labeled Corba. They represent a communication via Corba objects.

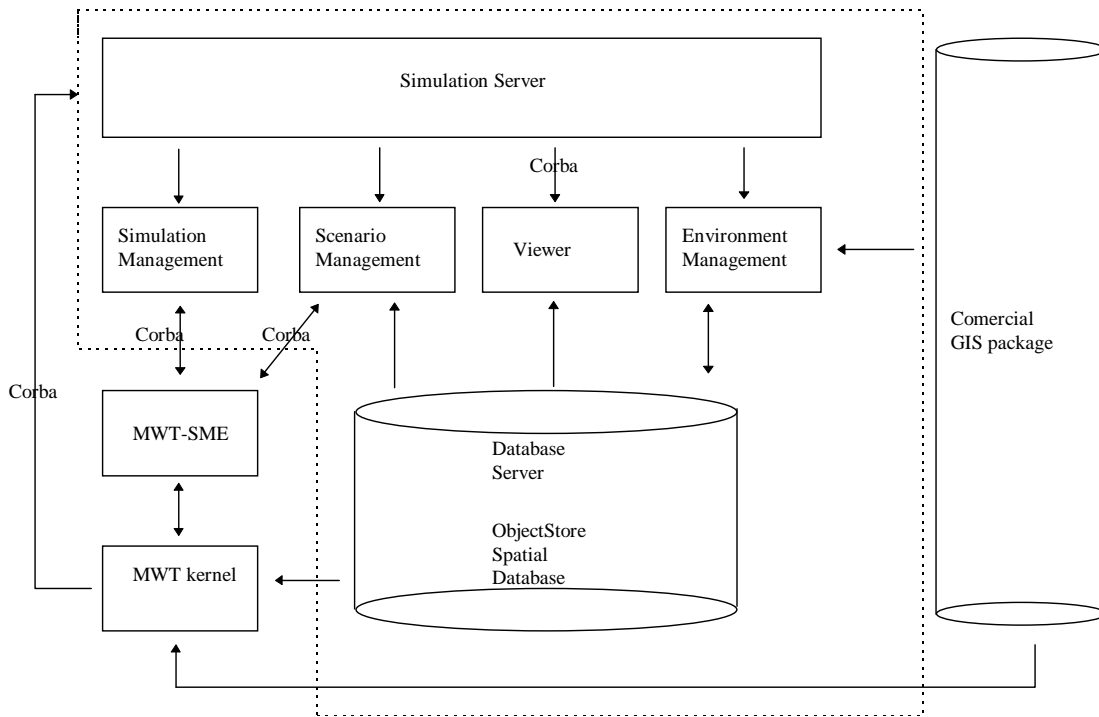


Figure 5. Modules contained in the MWT-GIS

4.3.2 Database

This section describes the structure of the database and the concepts of it. Firstly the idea of a parameter will be explained. Secondly the concept of maps is introduced. thirdly a description of the concept of environment will be given. Finally something is said about the database server.

4.3.2.1 Parameter

One of the requirements is that it should be possible to use parameters like SVP, water temperature and current and pH and that other parameters (not known or not important now) could also be incorporated in the MWT-GIS. After analysing the parameters and the structure of these parameters it became clear that parameters were build of the following components.

- One value. This could be a float, integer or string.
- Two related values. This could be a pair of floats or integers (For example current. It has a heading and a strength).

- A list of related values. For example on one position temperature as function of depth.

However, new parameters could have a different structure. One of the requirements was that the access time to retrieve data from the database should be low. Another requirement was that it should be easy to put new parameters in the database. These requirements can contradict each other. Low access time means simple database elements. Creating new parameters means complex data elements. To cope with these requirements, the concept of components was introduced:

definition: A component can be an elementary component (float, string or integer), or a list of other components.

definition: A parameter is a list of components with a description of each component.

An example of a parameter can be current. A current has a strength (a float) and a heading (a integer between 0 and 360). Figure 6 shows this parameter.

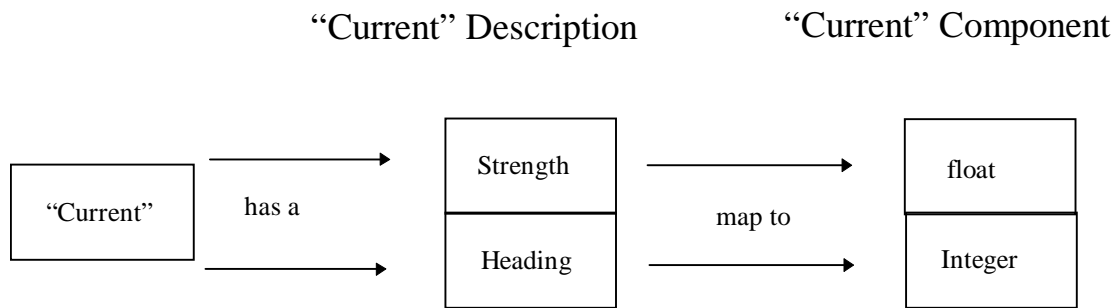


Figure 6. A parameter using components.

This component can be extended with a depth (float). A new component can be build by using a simpler component that is extended with depth. Figure 7 shows this concept.

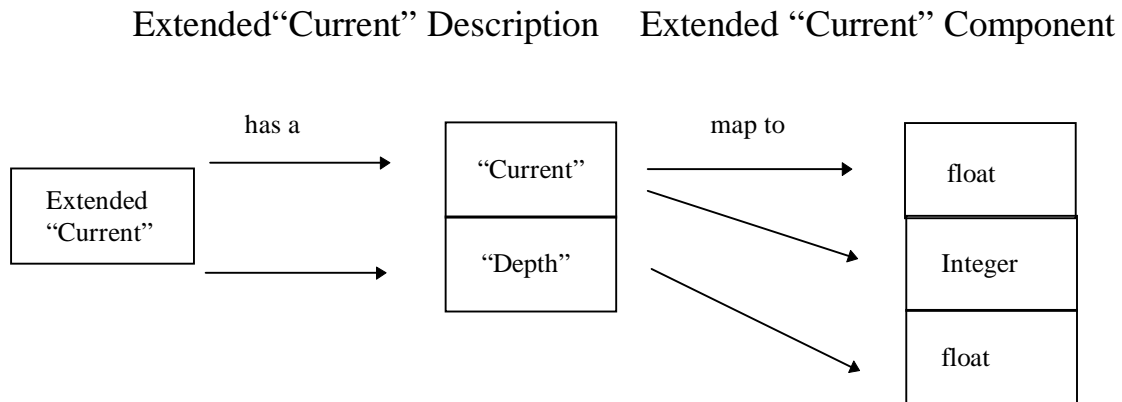


Figure 7. Extending a component.

It is now possible to define a more complex component: Column Current. This parameter describes the current on a position on various depths. This parameter can be build using several extended current component.. Figure 8 shows this parameter using the complex component of current. Using components results in several advantages:

- The components can describe every parameter that is described in the user requirements [Ling1].
- Components can be used to describe parameters that can be used in the future.
- Parameters can be made more complex (if needed) by extending the number of elementary components.

- Using elementary components will keep access time of the database low.

Users are now able to define components and parameters. Furthermore, it is the responsibility of the users to define logical and meaningful components and parameters.

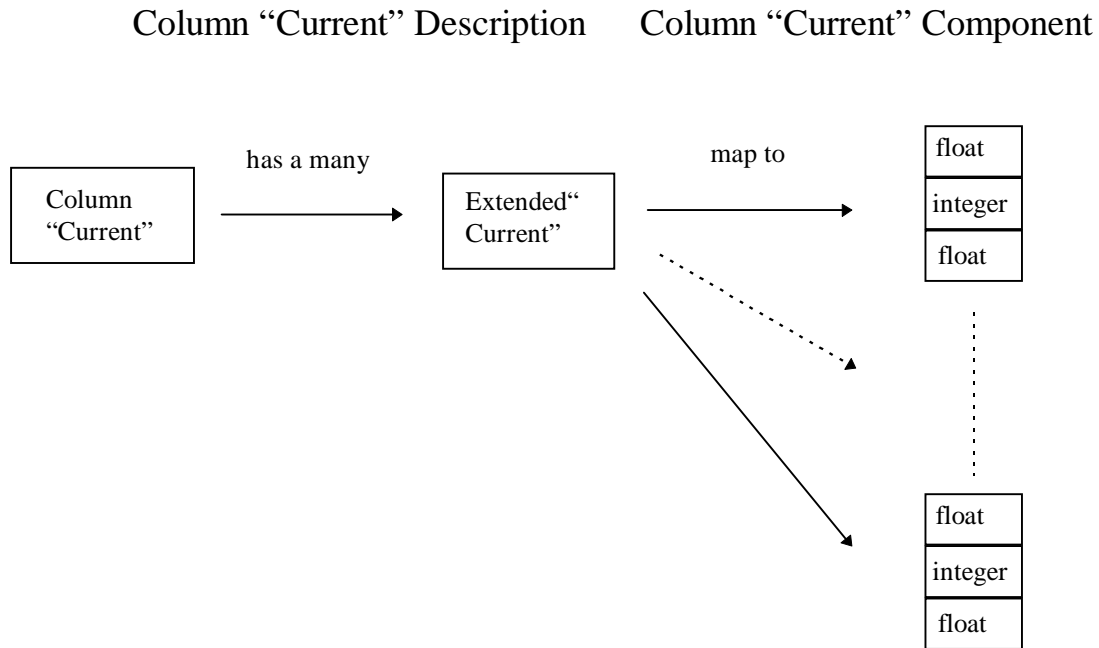


Figure 8 Parameter using a complex component

4.3.2.2 Maps

After the structure of the parameters have been specified values can be assigned to these parameters. These values are position and time related. This means there has to be a structure that uniquely identifies parameter values in time and position and parameter type. By using maps, this problem can be solved. The following definitions are used for the concept of maps:

definition: Parameter value is a parameter without it’s description but with a value assigned to every component of this parameter.

definition: A map is a structure containing the following: A set of time stamps, a bounding box, a set of parameter values derived from the same parameter.

A map is used to model a parameter in the environment. In this environment there are parameters that describe a surface (e.g. depth or current), lines (e.g. routes) or points (e.g. ship wrecks or mines). To model this a difference is made between these three maps. This leads to the following extension of the concept of maps:

definition: A surface map is a map such that for every position in the bounding box a parameter value can be returned using the set assigned to this map.

definition: A line map is a map such that every parameter value of the set is defined by a set of polylines.

definition: A point map is a map such that every parameter value of the set is uniquely defined by a position.

A map can contain point, line or surface data. Point or line maps can use one structure to store data. But surface data can be stored in different ways. For example, it is possible to divide an area into smaller areas using a grid, such that every grid box has a parameter value assigned to it. But also arbitrary polygons can be used. Besides these structures, numerous other structures can be used. Figure 9 shows three of these structures.

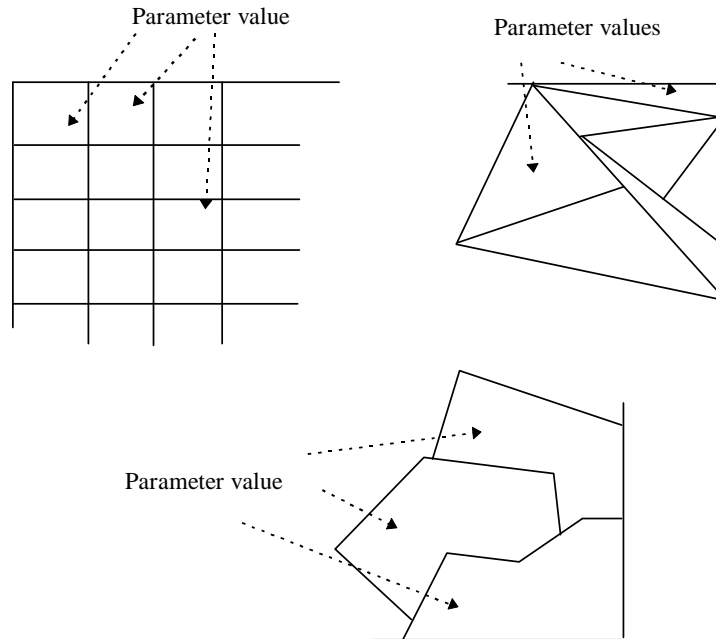


Figure 9. Grid, triangle, and a polygon structure.

Because different structures can be used for surface maps, it was decided to implement one structure (grid), but make a design, such that it is easy to extend it with other structures.

4.3.2.3 Environment

Using the concepts of parameters, components, parameter values and maps the definition of an environment can be stated:

definition: An environment is a collection of maps.

This environment is used to model the real world. For example, Table 1 is a model of the ocean and can be stored in this database structure.

Parameter	Format	Map	Time stamps
current	integer (heading) float (strength)	surface map (grid)	0:00, 6:00, 12:00, 18:00
depth	float (depth)	surface map (polygon)	0:00, 12:00
mines	string (type) float (weight) string (explosive type)	point map	0:00
ship routes	string (type) integer (width)	line map	0:00

Table 1. Model of the ocean.

Thus by using elementary building blocks parameters can be defined. Using the parameter values, maps can be defined. Maps are used to define an environment. Figure 10 shows this structure.

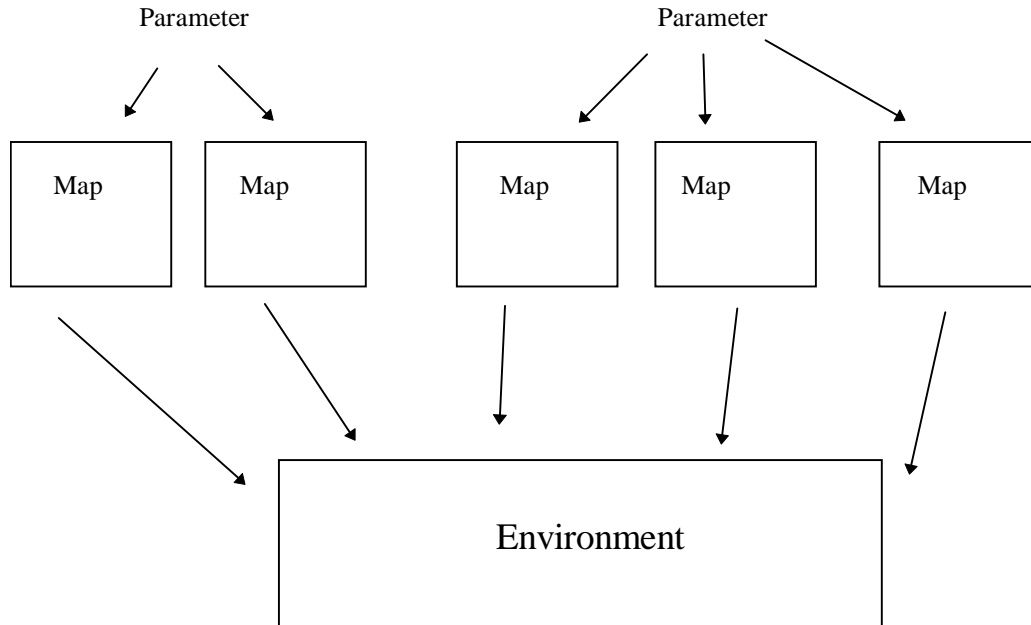


Figure 10. Schematic structure of database.

The structure described, is one of the many to model the real world. Because there was no literature on this subject except [Cons97] it is hard to tell if this is the best way to model the world in a database. The main reason for such a general spatial database were the unclear requirements. Users could not specify clearly what should be stored in this database. During interviews it became also clear that in the near future different environmental parameters could be stored in this database. Furthermore, it was not clear how to transform measurements of environmental parameters into realistic maps for a simulation. Because of this, it was decided to build a general model/database that can be extended and adapted in the future.

4.3.2.4 Database server

The database server is used to access, insert and modify data in the database. Because *Objectstore* has no query language, queries and operations on the database have to be defined. The tasks of the database server are:

- Retrieving data from the database.
- Retrieving meta data¹ from the database.
- Insert, delete and modify data.

The last task has the restriction that there can be only write access to the database by one process. The other tasks have multiple read access but no write access.

During a simulation, data can be needed in several different forms. This data can be acquired by either entities or observers. Entities participate in a simulation and usually represent a real life object (for example a ship or a sonar). Entities have a state and a concept of time. Observers monitor the simulation and can change the state of entities. Furthermore, observers produce output data of the simulation. This leads to the following queries in a simulation:

- An entity/observer wants to know the value of a parameter on a certain position.
- An entity/observer wants to know values of a parameter on a straight line. This can be useful for observers of the simulation that produce graphs of the environment.

¹ Meta data gives a description about the data contained in the database in a predefined format.

Figure 11 shows an example. An entity (ship) wants to know the depth beneath the ship. This ship also has a sonar. A sonar looks ahead using a sonar bundle thus scanning several metres in front of the ship. Furthermore, the simulation contains a sonar observer that displays the depth at the position the sonar bundle scans.

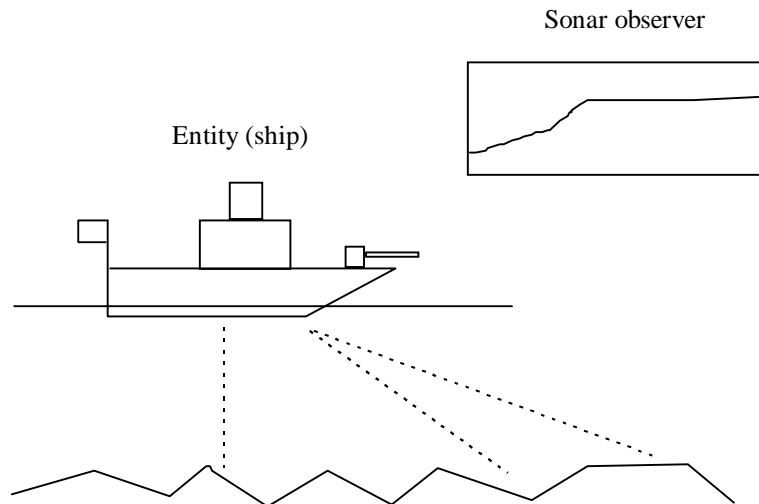


Figure 11. Entity and observer.

A viewer displays the simulation entities and the environment. To achieve this, a viewer queries the database for environment information. Because a viewer displays a region, it has to be able to perform region queries on this database. The previous discussion leads to the following operations the database server can perform:

operation	attributes	result
Point query	position, parameter, time	parameter value
Line query	2 positions, parameter, time, stepsize	set of parameter values
Region query	bounding box, parameter, time	a region of parameter values
Retrieve all meta data	none	information about all parameters and their format
Retrieve first meta data	none	information about the first parameter and its format.
Retrieve next meta data	none	information about the next parameter and its format.
Insert	bounding box, parameter values, time	map inserted
Delete	bounding box, parameter values, time	map deleted.

Table 2. Database server operations

4.3.3 Software requirements

During the system requirements some software requirements were formulated. This section lists the most important ones

- *Objectstore*. *Objectstore* is the database builder that will be used for the runtime database.
- *Orbix*. Corba implementations will be done in *Orbix*
- *Ilog*. *Ilog views* will be used for graphical user interfaces.
- *Sun*. The application will be developed on a Sun Sparc Ultra

- Read/write permissions. During a simulation or scenario definition the database can only be opened as read only. Only in the environment management module the database can be modified.
- Data format. Data will be delivered in ASCII format.

4.4 Conclusions

The result of the requirements phase was the system requirements document. This document describes the software requirements, constraints and structure of the MWT-GIS. The main problem was the lack of information about this subject. To overcome this, effort was put in gathering information of related areas such as geographical databases and geographical information systems. Furthermore, discussions with users led to a better idea about the MWT-GIS. During these discussions it became clear that there was also a need for other GIS functionality not related to the MWT-GIS.

5. Design

The design phase was the third step in developing the MWT-GIS. In this phase global and detailed design for the MWT-GIS were developed.

5.1 Objective

The primary goal of the design phase was to define the architecture and to specify the object structure of the MWT-GIS. While the System Requirements provides inside at a conceptual level, the design reflects the technical view of the designer with respect to these system requirements.

A secondary goal of the design phase was to identify and analyse technically difficult or critical parts of the design. Identifying these parts can lead to fewer implementation problems.

5.2 Method

5.2.1 Approach

The first step in realising the primary goal of the design phase was to identify use cases and scenario's related to these use cases. The use cases and scenario's are related to the user requirements and the system requirements. The second step was to draw object diagrams that are related to the scenario's. Using these initial object diagrams a conceptual design could be developed. Furthermore an architectural design could be developed. After that the object diagrams could be refined further. This resulted in the implementation design. For the design of the user interface part a graphical tool was used (*Ilog views*). Figure 12 shows this design approach.

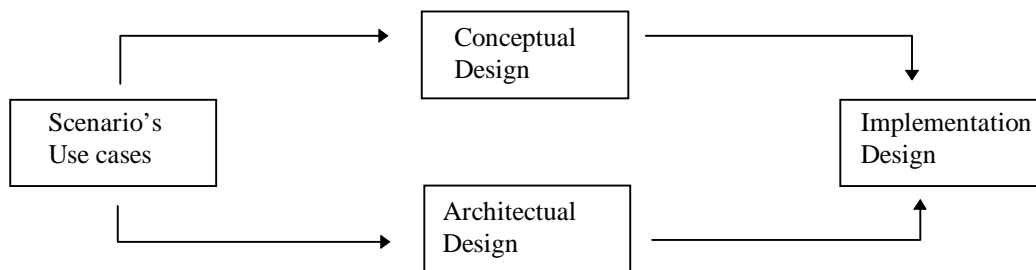


Figure 12. Design approach.

5.2.2 Problems

During the design phase several design problems were encountered. The first problem was how to model an environment in an Object Oriented environment. There was no literature on this. Furthermore, there was no literature about linking an environmental database to a simulation environment and combining an environment and simulation in a graphic display. Despite this lack of information there were several articles about GIS and Object Oriented databases. Furthermore, the article about the OpenGIS standard [Cons97] proved helpful in the design. The second problem was building a user interface. The design must be in such a way that it is easy to work with. To support this several GIS user interfaces were analysed to build a user interface that is easy to use. Because MWT-GIS is part of the MWT and can be linked to the MWT-SME effort was put in making it the same look and feel as the MWTSME interface.

5.2.3 Deliverables

The final result of the design phase was the MWT-GIS design document ([Ling3]) and a prototype of the user interface that can be used in the actual implementation. The MWT-GIS design document contains the following:

- Description of use cases and scenario's

- Description of the architectural model.
- Detailed design (object diagrams) of the MWT-GIS.
- Description of the user interface.
- Deployment diagram of the system.

5.3 Results

This section describes the design of the MWT-GIS. First, the architectural design is discussed. After that the detailed design will be discussed. Furthermore, a short description of the user interface will be given.

5.3.1 Architectural design

The architectural design gives a high level view of the design. It divides the system into smaller components and if necessary, these components are also divided into components. There are four main components. These components are independent applications that interact with each other.

- Database server. Contains the database and enables applications to query and modify this database.
- Simulation server. If the user starts the MWT-GIS this application will be executed. It will enable the user to start a simulation, define a scenario, modify the database or bring up a viewer.
- Viewer. This application can be used during a simulation run. It will display the environment and the entities in it. A user can visualise different entities and environment variables.
- Scenario management. Enables the user to define a new scenario, using a selected environment.

Figure 13 shows the high level architectural design. The components contained within the dotted line, represent the components discussed above. If we compare it with Figure 5, there is much resemblance between these figures. However, several modules are put together in Figure 13. The reason for this is that these modules do not contain much functionality.

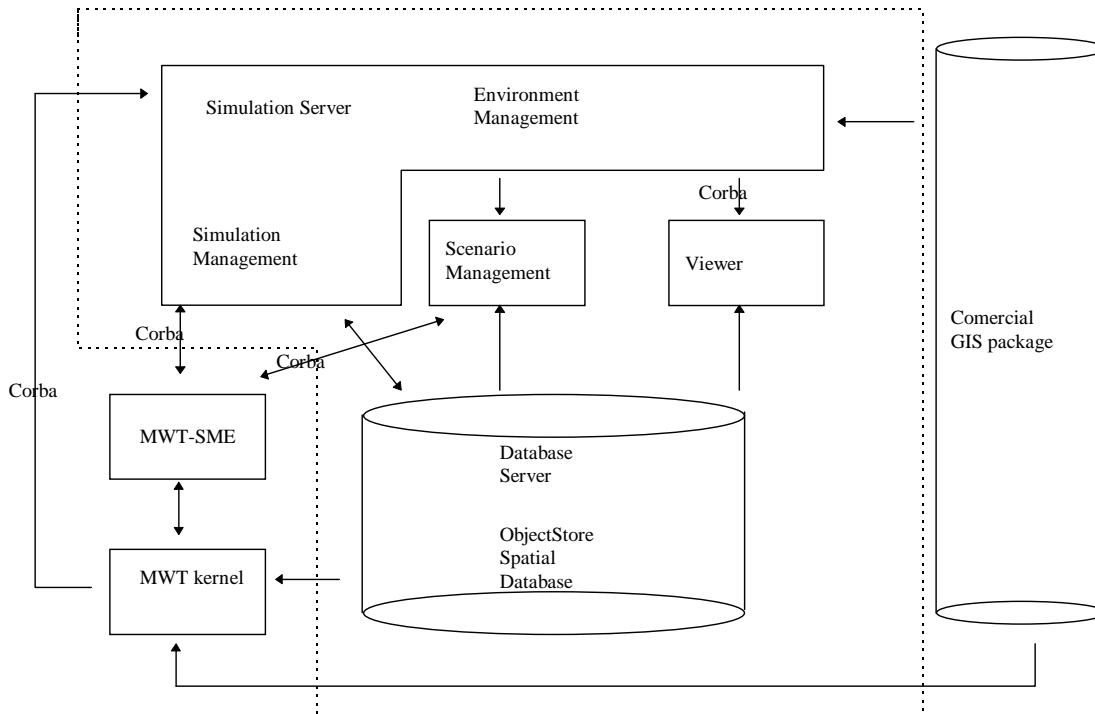


Figure 13. High level architectural design.

The high level design contains four applications. These applications have a (user) interface, communication to other applications and some additional functionality that performs actions initiated from the (user)

interface, or communication. The ANSI-Sparc model can be used to describe the architectural design of these applications. The ANSI-Sparc model is an architectural pattern that strongly resembles the PAC (Presentation, Abstraction, Control) architectural pattern (see [Bush96+]). The ANSI-Sparc model divides an application into three subsystems and a controller. These subsystems are:

- Interaction. Takes care of interactions between other programs or users. The application receives requests and returns the appropriate result. This subsystem can contain a (G)UI or an API.
- Delivery. This subsystem is used for storage of data or communication of data to other applications. A database can be part of this subsystem.
- Regulator. Interprets the actions performed in the interactor and performs certain manipulations related to these actions. For example an algorithm that has parameters as input (from interactor subsystem) and stores the output in a database (to the delivery subsystem).

These subsystems communicate by using events that are handled by the controller. Figure 14 shows the ANSI-Sparc model. The events are only send to the controller. The controller activates the proper subsystem.

The advantage of such a division of an application is modularity. These three subsystems can be build separately from each other if the interface is known in advance. This makes it easy to change a subsystem (e.g. another user interface).

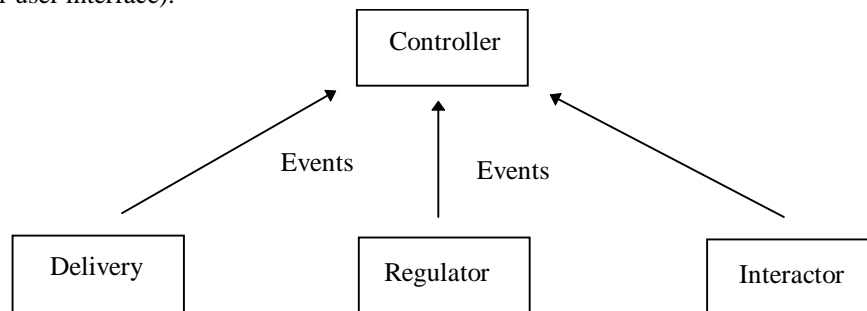


Figure 14. The ANSI-Sparc model.

5.3.2 Detailed design

This section discusses the most important object models of the MWT-GIS. A more detailed discussion can be found in [Ling3]. This sections discusses the database object model, the ANSI-sparc model of the simulation server and part of the user interface of the viewer.

5.3.2.1 Database

The database contains data from the natural environment. Thus an object model is needed to store this data, and to model this environment. First a description of the model will be given: An environment consists of different parameters. These parameters have a certain structure and can change in time and place. The object model contains the object “Environment”. This environment object contains several data layers (parameters). These layers contain several maps. The maps model the change of the parameters through time. Maps can have different times stamps to model this. Each layer has a relation with a value description object. This object contains a description of the structure of the parameter related to this layer. It is used as meta data to retrieve information from the database. Figure 15 shows the object model.

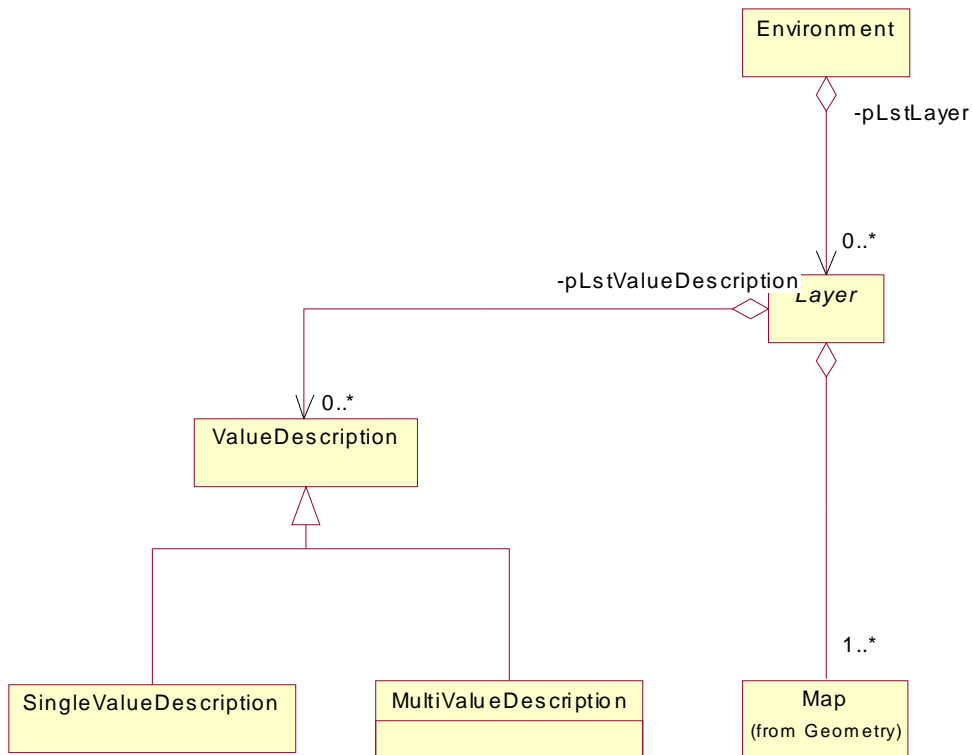


Figure 15. Environment object model.

Maps contain different types of spatial objects: 0 dimensional (points), 1 dimensional (curves), 2 dimensional (surfaces). A map contains only 1 dimension (a set of points, lines, or surfaces). This is consistent with the concept that a layer is related to 1 parameter. A parameter has only 1 dimension. Thus maps containing values of this parameter also have 1 dimension.

The structure of surface maps can be different in many ways. A map can contain a regular grid in which values represent a box, or a map contains only triangular polygons. In general, a set of surfaces is a set of closed polygons. However, if you have a special set of polygons (for example a grid), you can store and access this more efficiently. Figure 16 shows the object model of the maps.

A geometry object can be either a surface (2 dimensional), curve (1 dimensional), point (0 dimensional), or a map. A map can be a point map (set of points), curve map (set of curves), or a surface map (set of surfaces). The general surface, curve, and point concepts can be subdivided in specialised cases (like filled polygon and point2d). A surface map can be subdivided into special maps (for example filled regular grid and filled polygon). The advantage of the structure in is that it can be extended with new concepts of points, curves and surface, but also with new types of geometry's that have a different dimension.

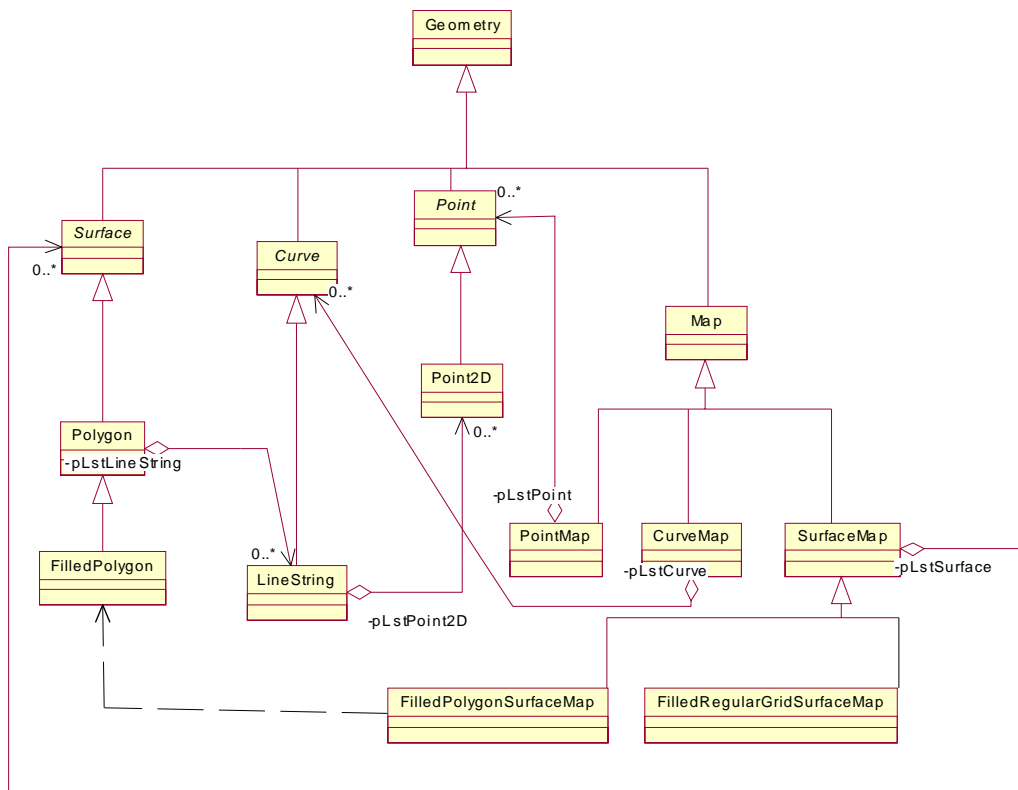


Figure 16. Geometry object model

The value package contains a structure for storing values in the database. During the user requirements phase it became clear that users did not know exactly what would be stored in the database. Furthermore, it was not clear what would be stored in the future. The structure of the parameters users want to store consisted of: float, integer, string or a combination of these three. To accommodate this structure, the composite pattern was used. The composite pattern not a complex pattern, yet its structure enables future users to define more complex values. Figure 17 shows this pattern.

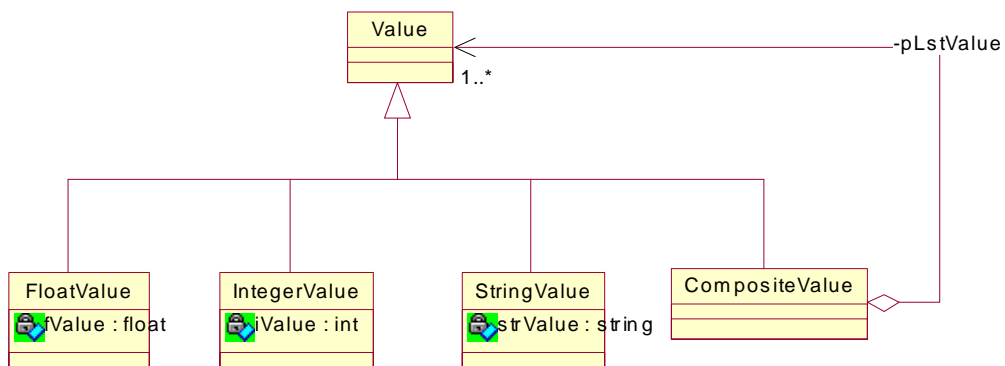


Figure 17. Value object model

5.3.2.2 Simulation server

This section will discuss the ANSI-Sparc model as it is used in the simulation server. The controller (simulation server object) has links to the three different modules. The controller contains a process event method. This method is called upon by the three different modules if they generate an event. The event mechanism is not concurrent. The SsInterface object is linked to the user interface of the simulation server and to the controller. If a user performs an action a method of this object will be called. For example the action start simulation method will start a simulation. This object sends an event to the controller that will process it. By disconnecting the user interface and its functionality it is easy to extend or change this interface in the future. The SsDelivery object is linked to the controller and the simulation. Furthermore, it has a list of links to delivery objects of viewers. A simulation sends information about entities to this object and this object distributes it to the different viewers.

The SsRegulator takes care of the action performed by the users. It manages the database and keeps track of paths and data files that are used. It has a link to the parser and to the controller. The SsRegulator object also contains a list of entities and observers participating in the simulation. It keeps track of the changes of these entities and observers. If a new viewer is activated this information is send to this viewer via the SsDelivery object. A viewer address buffer is used to send updates of entities and observers to the viewers.

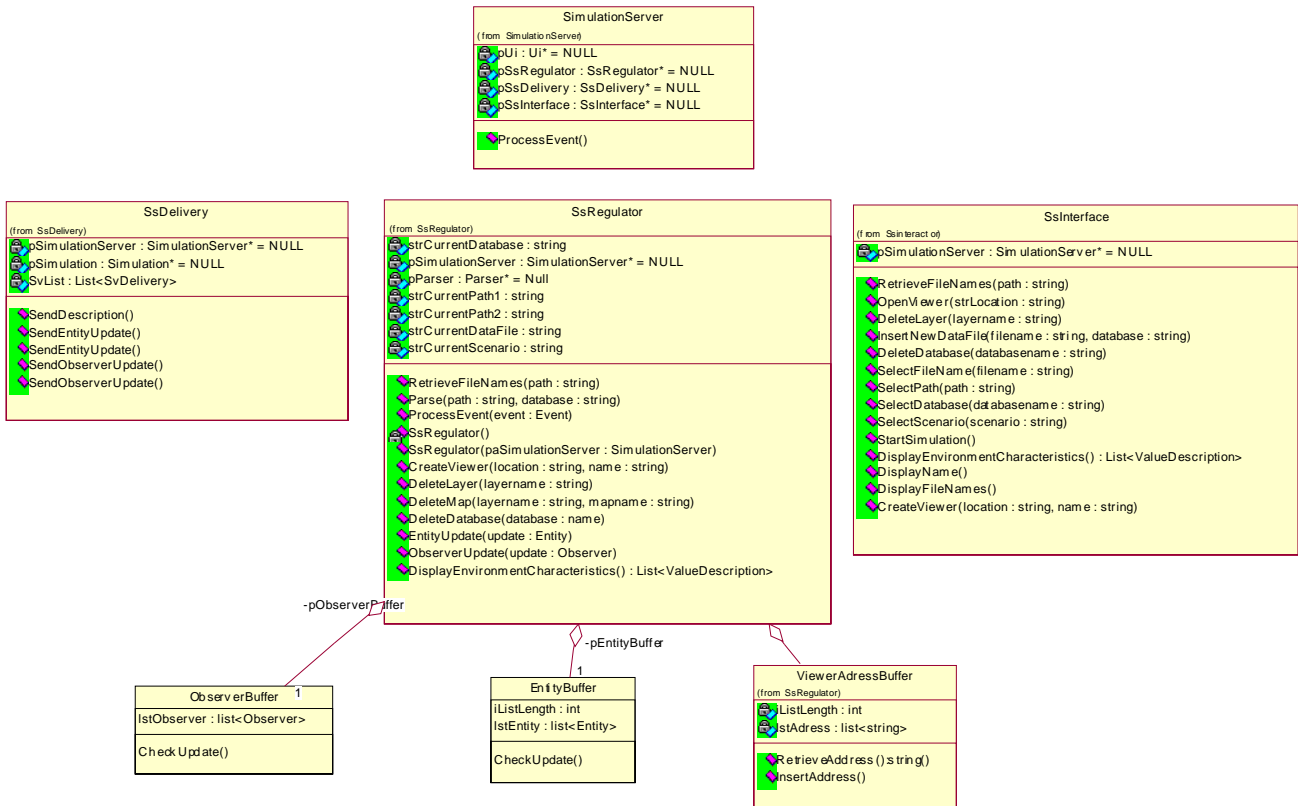


Figure 18. ANSI-Sparc model for simulation server.

The SsRegulator contains a link to the parser object. The parser object is used to parse data files and store this into the database. A general parser pattern was reused for this. This pattern is discussed in [Ling3] and [Smak95+]. This pattern can be used to build a grammar for parsing. Only the grammar rules and specific actions need to be defined. The grammar rules definition takes place in the parser object. Figure 19 shows this object. Rules can be either terminals, sequence rules, alternation rules, or repetition rule. The grammar is described in [Ling2].

```

Parser
pEnvironment : SequenceRule = new SequenceRule("ENVIRONMENT")
pPosition1_2D : SequenceRule = new SequenceRule("POSITION2D")
pPosition2_2D : SequenceRule = new SequenceRule("POSITION2D")
pLayers : RepetitionRule = new RepetitionRule("LAYERS")
pLayer : SequenceRule = new SequenceRule("LAYER")
pComponents : RepetitionRule = new RepetitionRule("COMPONENTS")
pComponent : AlternationRule = new AlternationRule("COMPONENT")
pSingleValue : AlternationRule = new AlternationRule("SINGLEVALUE")
pSingleValue1 : AlternationRule = new AlternationRule("SINGLEVALUE1")
pMultiValue : AlternationRule = new AlternationRule("MULTIVALUE")
pStringComponent : SequenceRule = new SequenceRule("STRINGCOMPONENT")
pFloatComponent : SequenceRule = new SequenceRule("FLOATCOMPONENT")
pIntegerComponent : SequenceRule = new SequenceRule("INTEGERCOMPONENT")
pPair : SequenceRule = new SequenceRule("PAIR")
pPairList : SequenceRule = new SequenceRule("PAIRLIST")
pMaps : RepetitionRule = new RepetitionRule("MAPS")
pMap : SequenceRule = new SequenceRule("MAP")
pDates : RepetitionRule = new RepetitionRule("DATES")
pMapType : AlternationRule = new AlternationRule("MAPTYPE")
pFilledPolygon : SequenceRule = new SequenceRule("FILLEDPOLYGON")
pFilledGrid : SequenceRule = new SequenceRule("FILLEDGRID")
pParametersParse : RepetitionRule = new RepetitionRule("PARAMETERSPARSE")
pPolygons : RepetitionRule = new RepetitionRule("POLYGONS")
pPolygon : SequenceRule = new SequenceRule("POLYGON")
pPositions : RepetitionRule = new RepetitionRule("POSITIONS")
pParameterParse : RepetitionRule = new RepetitionRule("PARAMETERPARSE")
pComponentParse : AlternationRule = new AlternationRule("COMPONENTPARSE")
pSingleValueParse : AlternationRule = new AlternationRule("SINGLEVALUEPARSE")
pSingleValueParse1 : AlternationRule = new AlternationRule("SINGLEVALUEPARSE1")
pMultiValueParse : AlternationRule = new AlternationRule("MULTIVALUEPARSE")
pPairParse : AlternationRule = new AlternationRule("PAIRPARSE")
pPairListParse : AlternationRule = new AlternationRule("PAIRLISTPARSE")
pMultiPairParse : RepetitionRule = new RepetitionRule("MULTIPAIRPARSE")
pBeginEnvironment : TerminalSymbol = new TerminalSymbol("beginenvironment", "begin(environment)")
pEnvironmentName : TerminalSymbol = new TerminalSymbol("environmentname", "environmentname=")
pEndEnvironment : TerminalSymbol = new TerminalSymbol("endenvironment", "end(environment)")
pBeginLayer : TerminalSymbol = new TerminalSymbol("beginlayer", "begin(layer)")
pLayerTypeString : TerminalSymbol = new TerminalSymbol("layertype", "layertype=")
pLayerName : TerminalSymbol = new TerminalSymbol("layername", "layername=")
pEndLayer : TerminalSymbol = new TerminalSymbol("endlayer", "end(layer)")
pBeginMaps : TerminalSymbol = new TerminalSymbol("beginmaps", "begin(maps)")
pEndMaps : TerminalSymbol = new TerminalSymbol("endmaps", "end(maps)")
pInteger : TerminalSymbol = new TerminalSymbol("integer", "integer")
pString : TerminalSymbol = new TerminalSymbol("string", "string")
pFloat : TerminalSymbol = new TerminalSymbol("float", "float")
pPairString : TerminalSymbol = new TerminalSymbol("pairstring", "pair")
pBracketOpen : TerminalSymbol = new TerminalSymbol("bracketopen", "{")
pBracketClose : TerminalSymbol = new TerminalSymbol("bracketclose", "}")
pPairListString : TerminalSymbol = new TerminalSymbol("pairliststring", "pairlist")
pBeginMap : TerminalSymbol = new TerminalSymbol("beginmap", "begin(map)")
pBeginDate : TerminalSymbol = new TerminalSymbol("begindate", "begin(date)")
pEndDate : TerminalSymbol = new TerminalSymbol("enddate", "end(date)")
pEndMap : TerminalSymbol = new TerminalSymbol("endmap", "end(map)")
pFilledPolygonString : TerminalSymbol = new TerminalSymbol("filledpolygon", "filledpolygon")
pGridSizeH : TerminalSymbol = new TerminalSymbol("gridsizeh", "gridsizeh=")
pGridSizeV : TerminalSymbol = new TerminalSymbol("gridsizev", "gridsizev=")
pHekje : TerminalSymbol = new TerminalSymbol("hekje", "#")
pStringParse : StringTerminal = new StringTerminal()
pStringParse1 : StringTerminal = new StringTerminal()
pStringParse2 : StringTerminal = new StringTerminal()
pStringParse3 : StringTerminal = new StringTerminal()
pFloatParse : FloatTerminal = new FloatTerminal()
pIntegerParse : IntegerTerminal = new IntegerTerminal()
pIntegerParse1 : IntegerTerminal = new IntegerTerminal()
pIntegerParse2 : IntegerTerminal = new IntegerTerminal()
pIntegerParse3 : IntegerTerminal = new IntegerTerminal()
pCreateEnvironmentAction : CreateEnvironmentAction = new CreateEnvironmentAction()
pCreateLayerAction : CreateLayerAction = new CreateLayerAction()
pGatherDateAction : GatherDateAction = new GatherDateAction()
pGatherComponentAction : GatherComponentAction = new GatherComponentAction()
pCreateMapAction : CreateMapAction = new CreateMapAction()
pGatherPairAction : GatherPairAction = new GatherPairAction()
pCreateValueAction : CreateValueAction = new CreateValueAction()
pGatherValueAction : GatherValueAction = new GatherValueAction()
pGatherPositionAction : GatherPositionAction = new GatherPositionAction()

Parser()
DefineRules()
Parser()
Parse(path:string,database:string)

```

Figure 19. Parser object

5.3.2.3 Viewer

When building a user interface with multiple windows a state manager is needed to manage the states of the windows. Ilog Views (the tool that is used to build these windows) has the option to define the states of the windows. Therefore this part will not be discussed. Figure20 shows an overview of the windows of the viewer user interface. The arrows indicate that a window can activate another window. Windows can be active at the same time. Thus if a user activates the entity list window via the viewer window, both windows are active. The entity attributes and observer windows can be made active several times, because there is more than one entity and observer and these windows are used for monitoring the simulation.

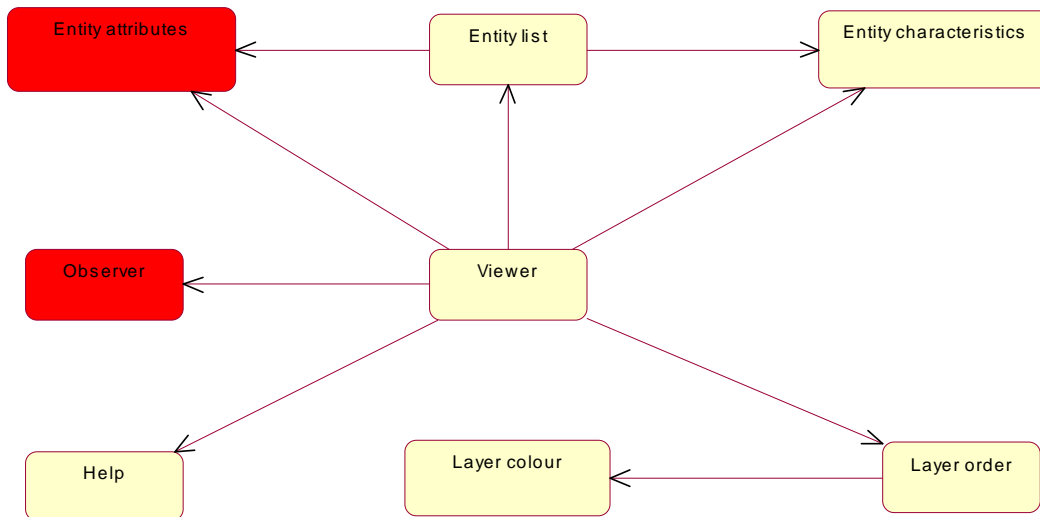


Figure20. Overview of different windows from viewer applications.

The viewer is the main window. It displays the environment and the simulation. The entity list window is used to select entities and change the characteristics of these entities (colour, visibility, and tracking) or to monitor the attributes of these entities via the entity attribute window. A user can activate different observers that display the change of certain parameters or attributes. Because of the complexity of the environment a user can adjust the colours of the different layers and the order in which they are displayed. The layer order window is used for this. The first column of this window display the dimension of the layer. A line is 1 dimensional, a dot 0 dimensional, and a surface 2 dimensional.

Figure 21 shows several windows of the viewer applications that can be active at the same time. The viewer window shows a simulation and its environment. Two entity attributes windows are active and two observer windows are active. The entity list window is active and can be used to activate more entity attribute windows or entity characteristics windows. Also the layer window is active. A user can select a layer and redefine the colour settings.

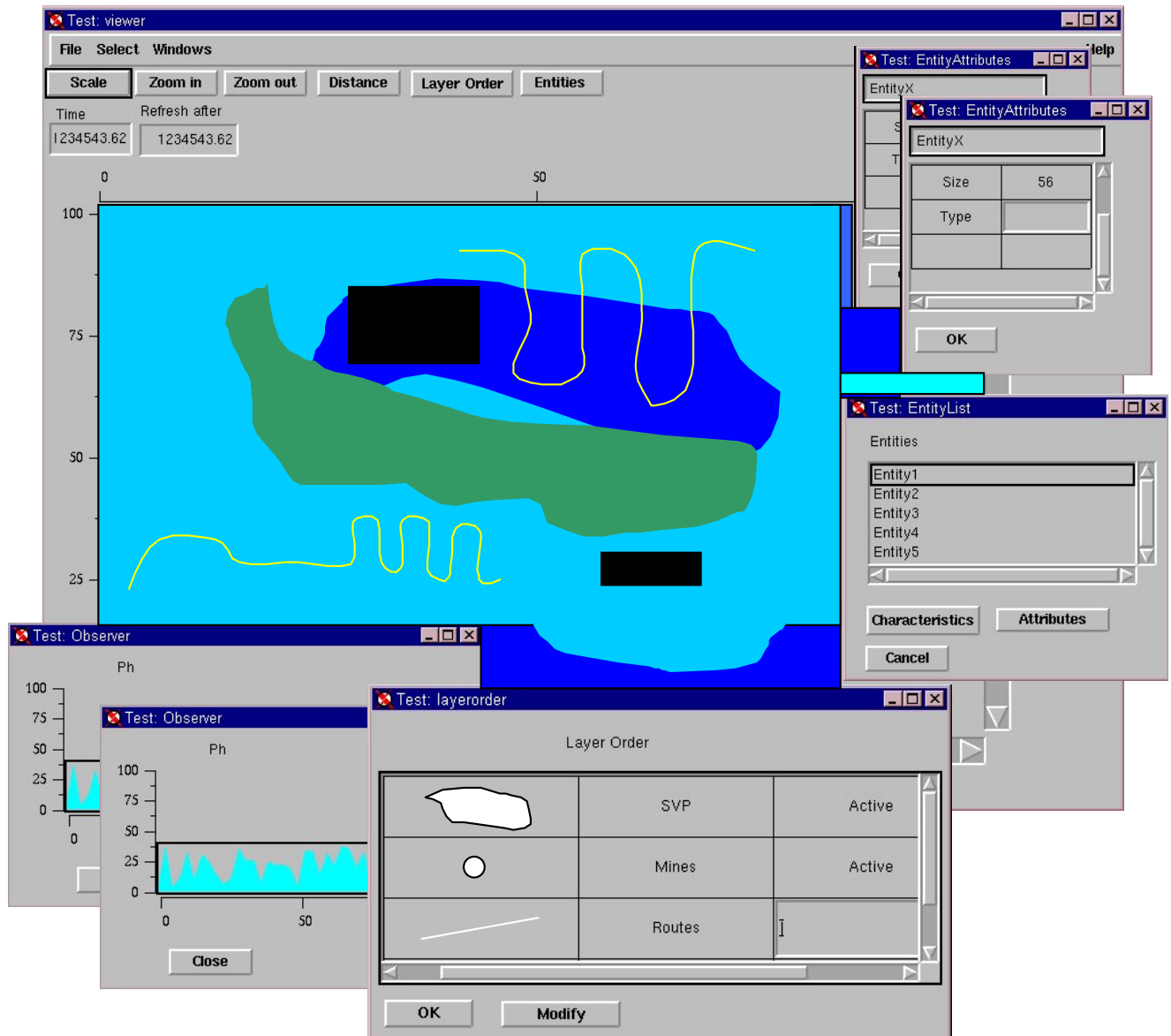


Figure 21. User interface of viewer.

5.3.3 Design decisions

During the design several design decisions were taken. The most important ones will be discussed.

Architectural design decisions

- **Ansi Sparc model.** The Ansi Sparc model provides a reference model for the low level architectural design. Because of the different components in this model (delivery, regulator, and interactor) it is relatively simple to change certain functionality without rewriting large parts of the code. The different components provide a separation of concern idea. Different components can be plugged in if necessary. A drawback of this model is the number of different events that need to build.
- **MWT-SME link.** The MWT-GIS enables a user to define a scenario and to select a scenario for simulation. The simulation kernel, and scenario editing/browse capabilities are already available in the MWT-SME.

- **Components.** The MWT-GIS contains a lot of functionality. It is logically to put this functionality into groups. Thus groups contain functionality that is strongly related to each other. The next logical step is to make components using this group functionality.
- **Distributed applications.** The MWT and MWT-SME is already a distributed application. The model database, kernel and browser can be used from different computers. This enables users to run simulations on faster computers and increases the flexibility of the application. The MWT-GIS contains different applications that can run on different computers. Again this increases the flexibility of the program. Furthermore, the database is located on one computer, it is not desirable to run everything on that computer. Two users may want to look at a simulation behind their own desk. To achieve this two viewers can be started on different computers.
- **Multiple viewers.** Multiple viewers enable multiple users to analyse a simulation. Furthermore, it enables one user to analyse a simulation from different views. The environment can consist of lots of layers. These can not be displayed in one viewer. Furthermore, a user might want to track different entities or observers. Sometimes two screens will be needed to accommodate this. Viewers can therefore be distributed on different computers.

Other design decisions

- **Reuse of parser pattern.** Reusing the parser pattern saves implementation time. Furthermore, it is a general parser pattern that can be easily adapted if the grammar changes.
- **Use of environmental storage pattern.** There was no information about how to store environmental information into an object oriented database. The only article about this was [Cons97]. This article also contained a class diagram for storing environmental information in an extended relational database. This diagram is used for the environmental storage pattern.
- **Ascii data files.** There are a lot of standards with respect to environmental information. One of the requirements was that data from the MWDC (Mine Warfare Data Centre) could be stored in this database. This data consisted of ascii files that contained measurements. Furthermore, ArcView and ArcInfo data files were used. It turned out that it was not yet clear what data TNO-FEL would get from the MWDC. Furthermore, it was not yet known what format would be used. This was the reason to use a ascii format for the MWT-GIS. This has several advantages. Ascii files are easy to read and to manipulate. If another standard or format would be used it is not difficult to write a converter.
- **Limited query capability.** The database has only limited query capability. New queries can not be constructed without rewriting the code. The reason for this is that Object Store does not contain a query builder. The programmer has to build the queries by programming. This would result in building a query engine. But building such an engine takes a lot of time. Therefore, it was decided to implement the queries that would be used in a simulation.
- **Database can contain different environmental structures.** It was not clear what geographical related parameters the database should contain. The database should be able to store water depth, SVP, and pH. But also other geographical related parameters. Because it was not known in advance what structure the geographical related parameters should have, it was decided to implement building blocks that could be used to store a wide range of these parameters.
- **Non concurrent event mechanism.** The events are processed in a serial manner. This is not a problem because the system will not have to cope with a lot of events (users will not perform thousands of actions per second).
- **Composite pattern for storing values.** During the user requirements phase it became clear that users did not know exactly what would be stored in the database. Furthermore, it was not clear what would be stored in the future. The structure of the parameters users want to store consisted of: float, integer, string or a combination of these three. To accommodate this structure, the composite pattern was used. The composite pattern not a complex pattern, yet its structure enables future users to define more complex values.
- **Reuse of graphical capabilities of observers.** The MWT already has some graphical functionality built into the observers. By using this functionality one can save time during implementation. Furthermore, one prevents the development of two different display techniques.

- **User interface of viewer resembles commercial GIS user interfaces.** Users will have to work (or already do) with other GIS packages. The ease of use will increase if the user interfaces of these packages resemble each other.
- **User interface of simulation server resembles the MWT-SME** Users of the MWT-GIS are also users of the MWT. To increase the ease of use and acceptance of the MWT-GIS the user interface resembles the MWT-SME user interface.

5.4 Conclusions

Rational Rose 5.0 was used in making the design. This tool is designed to keep an overview of the design and supports the UML language. It was sometimes difficult to keep an overview of the design because there are a lot of objects contained in the design. By using packages from Rational Rose and the ANSI-Sparc model this could be structured. The ANSI-Sparc model makes the design more transparent. A drawback of this model is the event mechanism. Although it makes it transparent the number of different events is high. These have to be designed and coded.

By using the OpenGIS consortium standard for ERDB as a reference we could model spatial objects for the MWT-GIS database. This standard gives a solid basis for defining new spatial objects. There are not many design patterns incorporated in the design. One of the reasons for this is the lack of experience with it and the limited time to experiment with it. Using design patterns could be beneficial both in time spent on design and implementation as well in reusability and extendibility.

For user interface design Ilog was used. This is a tool that graphically supports object oriented design and implementation of user interfaces. This saves time in designing user interfaces. Unfortunately our experience with Ilog was limited. Thus time saved on graphical design was spent on reading the manual.

6. Realisation

This chapter describes the realisation phase of the MWT-GIS project. In this phase the database and the simulation server were implemented. Also a beginning was made on the viewer. Because of time limitations the scenario manager could not be implemented.

6.1 Objective

The aims of the realisation phase were:

- Implementing parts of the database, simulation server and viewer, such that basic functionality could be used.
- Testing these parts.
- Giving presentations at TNO-FEL and TUE.

6.2 Method

6.2.1 Approach

Although the applications should run on a Unix environment, it was decided to develop the applications in Microsoft Visual Studio 5.0. The reason for this was that one of the team members had extensive experience with this tool and the C++ compiler under windows95. All source code and other documents needed for implementation were put under Microsoft Visual Source safe. Because Ilog developer studio was only available on a Unix environment, Ilog user interface files were generated on a Unix environment and ported to a Windows95 platform. Code (only the header files) was generated using the Rational Rose object design tool. To accomplish this, a Visual Basic script had to be adapted to generate header files specific for the MWT-GIS. Because Ilog and *Objectstore* were only used in a Unix environment, several libraries had to be installed. After this a proper directory structure had to be constructed such that it could be used for implementation.

It turned out that implementing the complete MWT-GIS applications in the given time would not be feasible. Therefore, it was decided to implement parts of applications such that they could be used but only with limited functionality. First the database server was implemented. However, the database only supports two dimensional parameters. The functionality of the simulation server is limited to inserting and deleting database items. Existing code was used for checking geographical data before inserting it in the database. A simple viewer was implemented for visualisation. This viewer could support displaying one layer and simulation objects.

The advantage of this approach, is that at the end of this project, applications could be used although with limited functionality. This enhances the involvement of the people working with it. Their feedback can be used in the implementation trajectory. Thus getting a better end product.

The other activities during the realisation phase were writing the final report and preparation of the presentations at TNO-FEL and the TUE. Writing the final report was spread out over almost the entire realisation phase, and presentations were held at the end of this phase.

6.2.2 Problems

During implementation, numerous problems were encountered. The most important ones will be discussed.

- Installing/Including libraries. Because development was done on a windows95 platform several libraries had to be installed. Also libraries had to be included. This took a lot of time because nobody had any experience with it.

- Reusing code. Code was reused that was developed on a Sun platform. This gave some difficulties in porting it to a Windows95 platform. Also some parts in this code had to be adapted because of this.
- Generating code using Ilog or Rational Rose. There was no experience with code generation and Ilog. It took a lot of time to find out how things were generated and how it could be used and adapted. Before code could be generated with Rational Rose a Visual Basic script had to be rewritten such that the generated code had the desired format.
- Non familiarity with Ilog and *Objectstore*. Because none of the team members had any experience with Ilog and *Objectstore* a lot of time was spend on getting familiar with these packages.

6.2.3 Deliverables

The results of the realisation phase were:

- Implementation of part of the database, simulation server and viewer.
- The final report for the OOTI course.
- Presentations given at TNO-FEL and the TUE.

6.3 Results

This section describes the implementation of the specific parts of the MWT-GIS. Due to the complexity of the MWT-GIS there was not enough time to implement the complete MWT-GIS. Only parts of the simulation server and database server are implemented. Also some comments about the reusability and extendibility of the implementation are given.

6.3.1 Database server

Rational Rose was used in generating header files from the UML design. Generating header files saved a lot of coding time. Another advantage was that header files were put into separate directories. Thus imposing a structure for coding of the classes. The ANSI-Sparcmodel also proved helpful. It enabled us to keep an overview of the code. A drawback of this model is the number of events that had to be implemented. Another problem that appeared during implementation was the indexing of data to be stored. To store and retrieve data efficiently a structure is needed to search this data (e.g. tree, matrix or grid). This structure is used to uniquely identify data by its index. Depending of the type of data, different index structures can be used.

Reusability and extendability were taken into account during the design phase. Because header files were generated from this design, reusability and extendability also became part of the implementation. The use of the design for building and storing spatial objects described in [Cons97] enables future programmers and designers to extend this design with new spatial structures. These types can be either 0,1 or 2 dimensional. It is also possible to introduce higher dimensional spatial objects. For example a structure containing space, time and speed and heading is 6 dimensional. The use of the ANSI-Sparc model makes it easy to extend new functionality or change existing one. The API, database, and manager can be changed indepently. By using new events or changing existing ones this functionality can be implemented.

6.3.2 Simulation server

The same method was used as with the implementation of the database server. Header files were generated and the ANSI-Sparc model was used as a architectural framework. Instead of an API a GUI had to be implemented. Ilog view was used to implement the GUI. This package enables a user to design a GUI and generate executable code. After that only the functionality behind the GUI needs to be implemented. This is not a difficult task, because the GUI and its functionality are separated in the ANSI-Sparc model. The parser that was needed for scanning new environmental data, was reused from the MWT. Only the grammar needed to be changed.

The reusable parser pattern enables future programmers and designers to adapt the grammar if nessecary. Furthermore, the ANSI-Sparc model is easy to extend as discussed in the previous section. The GUI is easy

to modify by using the Ilog package. By using the description of the old GUI stored in a script file, it can be modified for future needs.

6.4 Conclusions

Parts of the database server, simulation server and database were implemented. Unfortunately there was not enough time for the viewer and the scenario manager. The database can be used in simulations.

The unfamiliarity with *Objectstore* Ilog and Microsoft visual studio cost a lot of time. Also a lot of time was spent on installing libraries for *Objectstore* and Ilog. Reusing code can save a lot of time. But good documentation of the reused objects is necessary. Reengineering the reused code resulted in better insight in this code. Generating code from object diagrams can also save time and structure the implementation. However to do this, the design has to be complete and consistent. In this project only header files were generated. Depending on the completeness of the design, and the experience of the designer, it is recommended to generate code from a design tool and use round trip engineering. It saves time, and keeps the design and implementation consistent. Most design tools also have the possibility to put comments in objects. Using a script, a design document can be generated from this. Although this sounds as a good idea it is not widely used. There are several reasons for it. The first reason is that not everybody uses design tools and does not know what the possibilities are. The second reason is that people are using a design tool but never had time to gain insight in the possibilities of this tool. This is the reason why we only generated header files. The main lesson is: If you use a tool, be sure that you know what its capabilities are.

7. Conclusion

In this final chapter we evaluate the work we performed during the project and the approach used to specify and design of the MWT-GIS. We also give recommendations for future work on the MWTGIS.

7.1 Evaluation

In this section the project is evaluated. First, the phases and their end products are evaluated separately. Secondly, an overall evaluation is given.

7.1.1 Process

In this section, we evaluate per phase the approaches which were employed to develop the deliverables of this project.

- *Orientation.* This phase consisted of getting familiar with mine warfare, the MWT and get some knowledge about GIS. A lot was learned during this phase about mine warfare and GIS. It turned out that the MWT is a complex piece of software that had not yet been fully documented. Furthermore, we learned that a GIS is also a complex piece of software consisting of many components.
- *Requirements.* During this phase, requirements were discussed with the users. It turned out that users had no experience with a GIS and that it was difficult to formulate requirements. Requirements that were formulated, referred to standard GIS functionality rather than specific requirements for the MWT-GIS. After defining the user requirements, system requirements could be derived from it. Ilog views would be used for user interface design and *Objectstore* for database design. Because we were not familiar with these packages (although we had used them in the orientation phase), they were difficult to incorporate in the system requirements.
- *Design.* A large part of the design consisted of user interface design and object oriented design. No information could be found on building GIS user interfaces and object oriented geographically databases. Another problem was the lack of knowledge of Ilog and *Objectstore*. Some people might argue that a design should be independent of such commercial packages. This is not true. One can distinguish two kinds of designs, a conceptual design and a implementation design. The conceptual design is a high level design which is independent of any package used. It is a sketch for the implementation design. The implementation design is the actual design it represents what is going to be build. The impact of using libraries or packages should also be incorporated in the design apart from other constraints. Due to this, the design has undergone three major changes.
- *Preparation.* Because a lot of time was spent on the design there was less time for preparation. Preparation and Realisation merged together.
- *Realisation.* During this phase the design was implemented. Because of the size of the MWT-GIS not all parts could be implemented. It turned out that it was hard to plan the implementation. During implementation several designed objects had to be redesigned or extended. This aspect is hard to plan and time spend on this depends on the quality of the designer and programmer.

7.1.2 Overall

The MWT-GIS project was more complex than expected. There are several reasons for this complexity:

- It was difficult to establish user and system requirements. Future users found it difficult to define there requirements. Most requirements were requirements for a standard GIS. Besides this problem we had no prior experience with collecting user requirements and defining system requirements. Also there was no tool available to support collecting and managing user and system requirements.
- During implementation (and also design) of the MWTGIS project we had to use libraries of *Objectstore* and Ilog. It took a lot of time to familiarise with these packages. Even after that we only used 5% of the functionality simply because we did not know (and did not have the time) that there was another 95% of functionality.
- The size of the MWTGIS project was large. This adds to the complexity of the project.

During the project, the team was extended from one to two persons working on design and implementation. This made a great difference. One person could act as a sounding board for the other person. If one person makes a design it is easy to overlook some issues. Another person that also works on the projects can help prevent that. By discussing the design with each other it evolves to a better design. Questions like “Why do you do this?”, “What does that mean?” and “What do you think of this?” can make a difference in designing. It is important that the other person that is actively involved in the project, can see through any wrong line of thought. It is also important to have considerable knowledge about design patterns. They help in building a more simple, structured and reusable design. Equally important is architectural design. If the application resembles a standard problem, the same architectural structure can be used. Although design is separated from implementation, some problems can have language specific solutions. These “idioms” (see [Bush96+]) are the lowest level of pattern. Thus three levels of patterns can be distinguished (Figure22).

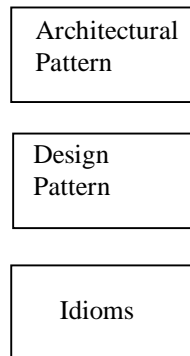


Figure22. Three levels of patterns.

Some people say that a good design can be made independently from the implementation. I tend to disagree with this. A good designer needs the experience of implementation confrontation. During implementation of a design, problems can appear that could be solved in the design or that can alter the design considerable. If this was foreseen during the design it could have saved time. One way to obtain this implementation confrontation is by implementing your own design. However, every new project has its own specific problems that come to surface during implementation. To overcome this problem a skeleton implementation can be build.

During the orientation phase one can already start building a skeleton application. This is an application that contains technology/libraries that will probably be used during implementation. Furthermore, it can be build in such a way it could serve as a skeleton for the final application. This application contains limited functionality but serves to tackle problems in an early stage. In this project such an application could be a window with a button that activates a graphical window and starts a simulation on this graphical window. This window shows some colours that are retrieved from a small database. Such a skeleton application contains Corba, Ilog and *Objectstore* components and interactions between them.

A skeleton application identifies problems during implementation that can be solved during the design. Furthermore, it can solve implementation specific problems such as the use of libraries.

A good design can be made independently from implementation but the best design is strongly related to its implementation.

7.2 Future work

Most of the design is finished, but not all is implemented yet. This is one of the tasks that has to be done in the near future.

Users want to use geographical information in their work. However the following considerations have to be taken into account:

- What geographical information do you want?
- Is this information available?
- What is the size of this data?
- How do you interpret geographical data? (can you interpret it?)
- How do you use different projections and co-ordinate systems?
- Do we need a geographical information system only for the MWT or can more projects benefit from this technology?

These questions are still unanswered and are difficult to answer if there is no experience with the possibilities of a geographical information system. Therefore it was difficult to formulate user requirements from the users point of view. I recommend the following steps in acquiring knowledge and experience about GIS such that in the near future this technology can be optimal exploited with respect to mine warfare and operations research:

- Start with a small commercial GIS package that can be extended
- Use this GIS in combination with the data that can be acquired
- Try to interpret and correlate these data sets.
- Try to find out what data would be desirable to have.
- Is this GIS package sufficient for the MWT? If not, can it be extended or replaced or do we have to build our own extension?

It should be noted that buying a GIS package is not the solution. A lot of time has to be spent in customising and structuring this tool.

Glossary

API	
ASCII	American Standard Code for Information Exchange II
ASW	Anti Submarine Warfare
AWW	Above Water Warfare
CORBA	Common Object Request Broker Architecture
EBF	Electronic Battlefield Facility
ERDB	Extended Relational Database
DLL	Dynamic Link Library
GIS	Geographical Information System
GUI	Graphical User Interface
Ilog	Tool for building graphical user interfaces
MCM	Mine Countermeasures
MW	Mine Warfare.
MWDC	Mine Warfare Data Center.
MWT	Mine Warfare Testbed
MWT-GIS	MWT Geographical Information System
MWT-SME	MWT Simulation Modeling Environment
<i>Objectstore</i>	Tool and libraries for building an object oriented database
OOTI	Ontwerpersopleiding Technische Informatica
Orbix	A full implementation of CORBA in C++.
SQL	Sequential Query Language
SVP	Sound Velocity Profile
TNO	Nederlandse organisatie voor Toegepast Natuurweten-schappelijk Onderzoek
TNO-FEL	TNO Fysisch Electronisch Laboratorium
TUE	Technische Universiteit Eindhoven
UML	Unified Modelling Language
UTM	

List of Figures

FIGURE 1 THE MARITIME OPERATIONS RESEARCH PYRAMID OF MODELS AND SYSTEMS.	7
FIGURE 2. PHASES IN RELATION WITH TIME.....	10
FIGURE 3 OVERVIEW OF THE MWT ARCHITECTURE.....	12
FIGURE 4. RELATIONS BETWEEN THE DIFFERENT MODULES	15
FIGURE 5. MODULES CONTAINED IN THE MWT-GIS.....	16
FIGURE 6. A PARAMETER USING COMPONENTS.....	17
FIGURE 7 PARAMETER USING A COMPLEX COMPONENT.....	18
FIGURE 8. GRID TRIANGLE, AND A POLYGON STRUCTURE.	19
FIGURE 9. SCHEMATIC STRUCTURE OF DATABASE.....	20
FIGURE 10. ENTITY AND OBSERVER.	21
FIGURE 11. DESIGN APPROACH.	23
FIGURE 12. HIGH LEVEL ARCHITECTURAL DESIGN.	24
FIGURE 13. THE ANSI-SPARC MODEL.....	25
FIGURE14. THREE LEVELS OF PATTERNS.	38

Bibliography

- [Bend1+] F.P.A. Benders and P.P.J. de Krom. *MWT Simulation Modelling Environment - Software Design Document*. Internal TNO-Report.
- [Bend96+] F.P.A. Benders and P.P.J. de Krom. *Design of the Mine Warfare Testbed Simulation Modeling Environment*, Eindhoven, Stan Ackermans Instituut, 1996
- [Bush96+] Frank Bushmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. *A System of Patterns (pattern oriented software architecture)*, Wiley, 1996
- [Cons97] Open GIS Consortium, Inc. *OpenGIS Simple Features Specification For SQL(Revision 0)*. 8 September 1997.
- [Gamm+] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns*. April 1996
- [Ling1] F.J.M. van Lingen. *MWT-GIS User Requirements Document*. Internal TNO report in preparation, 1998
- [Ling2] F.J.M. van Lingen. *MWT-GIS System Requirements*. Internal TNO report in preparation, 1998.
- [Ling3] F.J.M. van Lingen *MWT-GIS Design*. Internal TNO report in preperation, 1998.
- [Smak95+] C.H.J. Smakman and W.E.P. van der Sterren. *Design of the Mine Warfare Testbed*. Eindhoven, Stan Ackermans Instituut, 1995
- [Smak2+] C.H.J. Smakman and W.E.P. van der Sterren. *Mine Warfare Testbed - System Requirements Document*. Internal TNO-FEL Report
- [Noord] J. Noordam. *Link between MWT and RTPI*. TNO report FEL-96-S309, 1996.
- [Veld97] E.R. van Veldhoven. *Software Design Document*. Internal TNO report in preparation, 1997.