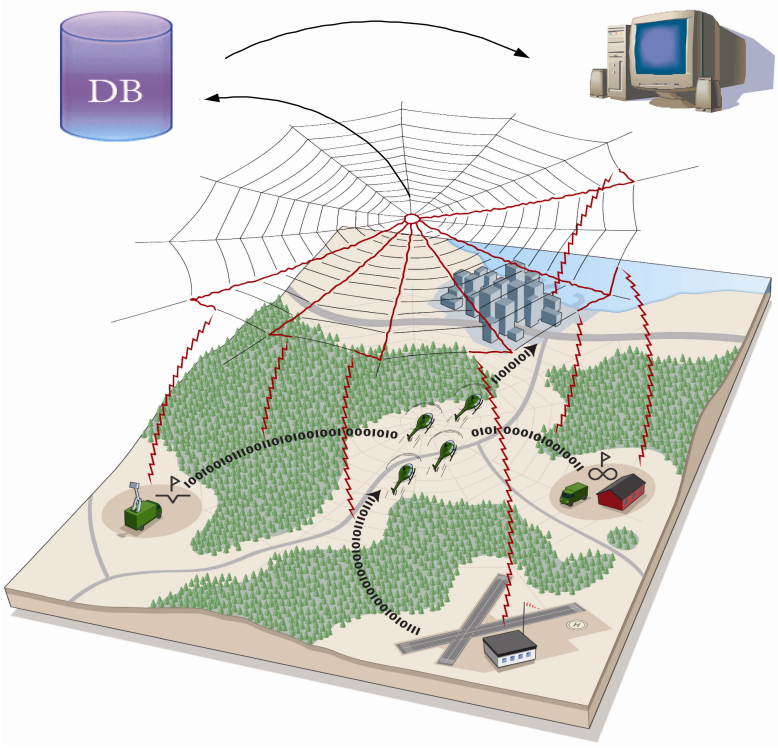
FOI-R--1277--SE
May 2004
ISSN 1650-1942
Technical report

Dennis Andersson
Christer Skagert

Managing Massive Datasets from Distributed Tactical Operations



Command and Control Systems
SE-581 11 Linköping

SWEDISH DEFENCE RESEARCH AGENCY

Command and Control Systems
P.O. Box 1165
SE-581 11 Linköping

FOI-R--1277--SE

May 2004

ISSN 1650-1942

Technical report

Dennis Andersson

Christer Skagert

Managing Massive Datasets from Distributed Tactical Operations

Acknowledgements

We would like to thank our examiner Prof. Henrik Eriksson, our supervisor Pär-Anders Albinsson and our project leader Mirko Thorstensson for their valuable support, comments and criticism throughout the entire project.

We would also like to thank Dr. Magnus Morin, Associate Prof. Johan Jenvald and Markus Axelsson from Visuell Systemteknik AB who assisted us with valuable comments and advisory during the project.

Finally we would like to thank Mattias Johansson and Per-Ola Lindell at the Department of Systems Engineering and IT Security at the Swedish Defense Research Agency in Linköping, Sweden, for their assistance with the project.

CHAPTER 1 INTRODUCTION 5

1.1 BACKGROUND 6
1.2 PROBLEM DESCRIPTION 9
1.3 APPROACH 10
1.4 STRUCTURE OF THIS REPORT 13

CHAPTER 2 DATA MODEL FOR STORING MISSION HISTORIES 15

2.1 RECONSTRUCTION AND EXPLORATION 15
2.2 MODELING THE DATABASE 16
2.3 BLOCK: MODELING DOMAIN 18
2.4 BLOCK: RECONSTRUCTION AND EXPLORATION 26

CHAPTER 3 SELECTION OF DATABASE MANAGEMENT SYSTEM 35

3.1 RELATIONAL DATABASE MANAGEMENT SYSTEMS 35
3.2 OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS 36
3.3 OBJECT-RELATIONAL DATABASE MANAGEMENT SYSTEMS 36
3.4 SELECTING CONCEPTUAL DATABASE MANAGEMENT SYSTEM 37
3.5 REQUIREMENTS ON THE DATABASE MANAGEMENT SYSTEM 37
3.6 IMPLEMENTATION OF THE DATABASE 41

CHAPTER 4 THE APPLICATION FRAMEWORK 45

4.1 DESIGN GOALS 45
4.2 DESIGN MODEL 47

CHAPTER 5 IMPLEMENTATION TECHNIQUES 51

5.1 PROGRAMMING ENVIRONMENT 51
5.2 INTEROPERABILITY 52
5.3 DATABASE INTEGRATION 53

CHAPTER 6 IMPLEMENTATION OF THE APPLICATION FRAMEWORK 55

6.1 REQUIREMENTS 55
6.2 DATA CENTRIC TIER 56
6.3 USER CENTRIC TIER 57
6.4 PRESENTATION TIER 58
6.5 TESTING APPLICATIONS 59

CHAPTER 7 APPLICATIONS **61**

7.1 GPS LOG IMPORTER **61**
7.2 MISSION SETUP TOOL **63**

CHAPTER 8 REVISIONS TO THE MIND FRAMEWORK **69**

CHAPTER 9 FUTURE WORK **73**

9.1 DATABASE **73**
9.2 APPLICATION FRAMEWORK **77**
9.3 APPLICATIONS **78**
9.4 MIND **80**

CHAPTER 10 SUMMARY AND CONCLUSIONS **83**

10.1 SUMMARY **83**
10.2 DATA MODEL **83**
10.3 DATABASE MANAGEMENT SYSTEM **83**
10.4 APPLICATION FRAMEWORK **84**
10.5 APPLICATION DEVELOPMENT **84**
10.6 GENERAL CONCLUSIONS **85**

CHAPTER 11 REFERENCES **87**

Chapter 1
Introduction

A Distributed Tactical Operation (DTO) involves a complex system of humans and artifacts that cooperate in a hierarchy to reach common goals with a given set of resources. To study DTOs we need to be able to deal with realistic environments – a valid context – to get insight into the socio-technical aspects of command and control. The Swedish Defense Research Agency presented a reconstruction-exploration approach for DTOs (Morin, Jenvald & Thorstensson, 2000) that aims at taking on the challenge of using a real-world context as the basis for DTO research. The approach comprises four principal activities needed to reconstruct the complex course of events from a DTO; domain analysis, modeling, instrumentation and data collection. The 'mission history' resulting from the reconstruction is then used for the exploration phase, where collected data are presented and made available for manipulation and analysis.

To handle and present the large amounts of data that are collected during field trials, the Swedish Defense Research Agency developed a computer supported framework called MIND (Morin, 2002a; Jenvald, 1999; Morin et al., 2000). The size of the mission history that holds all the collected data will vary depending on how detailed the conceptual models and instrumentation plans are. For large field trials the mission histories may become massive. Managing these massive datasets in the current solution which MIND is relying on now, a single file structure, may become an overwhelming task and a limiting bottleneck.

In this report we present a solution to store mission histories using a database management system. The solution has been explored using two problem domains based on military tactical operations and fire rescue operations. We also present a framework for building applications connected to the database in order to facilitate developing a distributed system consisting of several applications operating simultaneously on the same database. Finally we describe applications needed to operate the database for reconstruction and exploration, the applications we implemented and how we modified the MIND framework to utilize of the database.

We encountered some potential problems during the project which we discuss briefly. We also present a few suggestions on how to continue the work based on the insights gained during the project.

1.1 Background

Tactical operations are very complex and difficult to interpret. Since the environment is continuously shifting, no set of predefined rules will cover all situations. The decision maker must therefore rely on experience and personal judgment to make the best decisions based on what information is locally available and act according to it, even though the information is almost certainly incomplete, fragmented and sometimes even ambiguous. In many operations, military and rescue missions in particular, the consequences of an erroneous decision may be severe. (Morin, 2002a)

Needless to say, there is a great need for understanding the work involved in DTOs to develop improvements and support. However, understanding complex socio-technical work, like DTOs, is a difficult challenge. To take on this challenge the Swedish Defense Research Agency describe an approach for creating and analyzing a mission history based on the reconstruction and exploration of the complex course of events of a DTO (Morin et al., 2000), see figure 1. This approach can be used to get a better understanding of the particular situation and thereby get a better understanding of DTOs.

Reconstruction is described as the act of devising a conceptual model of an operational scenario and populating this model with data captured from an operation adhering to the scenario. The reconstruction phase consists of four distinguished steps: (1) domain analysis, (2) modeling, (3) instrumentation and (4) data collection. The mission history between the reconstruction and exploration steps is a composition of the conceptual model, the instrumentation plan and the collected data. Included in this history are data such as communication between actors in the operation, observation protocols created by observers, casualty reports, video clips and more. The exploration phase is described as the use of these multimedia models of tactical operations for reflection, discovery and analysis.

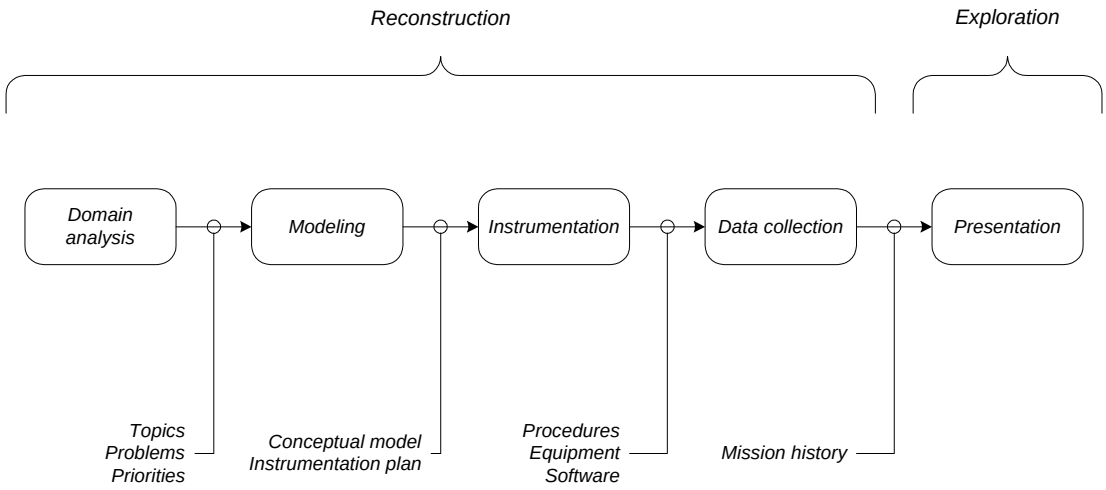


Figure 1 *Overview of the steps of the reconstruction-exploration approach. Boxes indicate the principal activities, whereas annotated arrows show the artifacts produced by each activity (Morin et al., 2000).*

To support the reconstruction of DTOs in field studies and the exploration of data with multimedia representations, MIND (Jenvald, 1999; Morin, 2002a) was implemented. MIND is a component-based analysis framework that integrates domain models, data sources, data converters and presentation views. MIND has been used to demonstrate the applicability of methods and tools in several different domains including combat training with the Swedish Army (Thorstensson, 2002a), naval operations with the Swedish Navy (Thorstensson, Jenvald & Morin, 2002) and Rescue operations with the Swedish Rescue Services Agency and Linköping Fire Rescue Department (Thorstensson, Björneberg, Tingland & Carelius, 2001; Thorstensson, 2002b).

Table 1 presents the different classes of components that the MIND framework distinguishes between. The objects are part of the conceptual model, the sources of the instrumentation plan while the events and documents are samples of collected data during the operation. Views and maps are used for exploration but have no pronounced meaning in the reconstruction phase.

The size of the mission history data will vary depending on how detailed the conceptual models and instrumentation plans are. Using a model where

Component type	Description	Examples
Objects	Objects model real-world elements of a taskforce. They can be organized hierarchically to model the structure and chain of command. State variables represent essential aspects such as location, capabilities, and resources.	Vehicles, Ships, Aircraft, People, Casualties
Events	Events represent time-stamped data collected during a tactical operation. Events define changes in object state variables at particular time points corresponding to time stamps.	Position sample, Observation report, Sensor sample
Sources	Sources manage collections of events from a particular physical or logical source. Sources are the primary mechanism for organizing and tracing data from an operation. Sources implement gateways and converters for accessing external data.	Picture source, Position source, Audio source
Views	Views are presentation windows for particular types of data. Customized views are the primary means of extending the presentation capabilities of MIND.	Map view, Casualty view, Audio clip view, Dynamic timeline, Attribute explorer view
Maps	Maps encapsulate a model of the earth, a projection method, and the logic necessary to render an image of this model in a generic map view.	Raster map, Vector map, Generic coordinate system view
Documents	Documents are static data, for example text, digital photographs, video clips, audio samples, local HTML pages, and Internet URLs. A document can be made dynamic by linking it to an activation event that specifies when it was created.	Text, HTML, Digital photograph, Video clip, Audio clip, URLs

Table 1 *Classification of the component types in the MIND framework (Morin, 2002b).*

100 vehicles are equipped with GPS receivers and logged every 10 seconds for 8 hours a day in 3 days will result in 70 MB of data assuming a coordinate can be stored as an 8 byte structure. This can be regarded as a relatively small amount of data, but what if we add physiological data logged every 10 milliseconds or continuous sound recordings from microphones placed on every soldier? It is not difficult to realize that the dataset may become massive.

In order to be able to recreate the events of a DTO in chronological order, all data for reconstruction must be time stamped. Obviously, time is an

important factor and how to handle it is a decisive aspect that needs to be carefully considered when creating a centralized storage facility for DTO data. Conventional database technology offers little support to deal with this kind of temporal data. Therefore an extensive research has been conducted during the last three decades in temporal databases (Jensen, 2000). These databases define several complex ways of dealing with time.

Another interesting research area somewhat related to this thesis is spatio-temporal databases and geographic information systems, GIS. The goal of the research in these areas is to reduce the interaction problems between highly advanced geographical data collection tools, like GPS-receivers and video cameras, and ordinary database management systems.

Our conclusion is that this project, though it involves a lot of temporal and spatial data, does not benefit at this early stage from the advanced modeling and design that is offered by temporal and spatio-temporal databases. Still, some areas of the research become interesting if further development of the data model is pursued.

1.2 Problem description

The conceptual model and instrumentation plan constructed during the modeling phase will decide how much information will be available at the exploration. In order to create a realistic reconstruction of the scenario the model must be correct and detailed enough to contain all necessary information. Deciding what is necessary is not trivial.

To prevent loss of potentially important data, the model often becomes large with many data sources recording large amounts of data. The dataset of a recorded mission may thus grow rapidly. The current implementation of MIND stores this dataset as serialized objects in a single file. The purpose of these files is to offer a way to save a composed mission history so that it may be reloaded at another session.

As the size of the dataset increases it gets more difficult to manage and the main purpose of this project is to investigate how a database management system can be used to overcome this problem. The current approach involves no central storage facility for collected data or conceptual models. Mission histories are composed out of conceptual models created using the MIND framework by manually importing scattered log files of data. Since the conceptual model, instrumentation plan and the data collection are

stored together it is difficult to reuse concepts, plans or data used in previous missions.

The following questions have been formulated for this thesis:

- How can a general model be designed, that can hold mission histories from DTOs, including conceptual models, instrumentation plans and collected data?

- What demands does a general data model for storing mission histories for DTOs put on a database management system and which database management systems are available that can handle these requirements?

- How can a general application framework be developed which may be used to build applications operating on the data in a mission history database?

- What applications are necessary for the database to support reconstruction and exploration of DTOs?

1.3 Approach

This project involves several different tasks, listed as main thesis questions in section 1.2 above. Some of these tasks are closely related to each other and could be solved using the same approach, while some are completely different and therefore need specialized approaches. The approaches are presented in the same chronological order as the corresponding problems were undertaken, see figure 2.

1. The first phase of the project consists of defining the scope of the project and after this we started to develop a general data model for storing mission histories trying to answer the first two main questions.

2. The second phase, the analysis of database management systems, is related to the third and fourth question in the previous section. The last two phases are directly related to the development of application framework and applications slot.

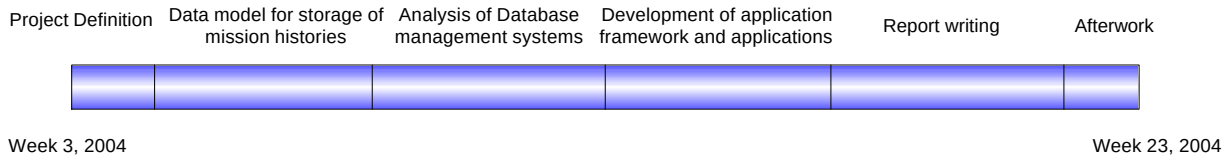


Figure 2 *Project timeline describing the different phases of this thesis project. Note that the time slots given to each phase in this timeline are not proportional to reality.*

3. The third and final phase consists of the two last slots of the timeline, which deal with the development of this report and the afterwork needed to finish the project.

1.3.1 Development of a data model for storing mission histories

The approach we used for the development of the data model for storing mission histories in DTOs is loosely based on the spiral model (Boehm, 1986), which is an iterative software life-cycle model suited for object oriented development. Every step in our version of the spiral model consists of analysis, design, implementation and testing (see figure 3). We used this model to find weaknesses and strengths in an early stage so that the model could be refined until we reached a satisfying model.

We found this approach very useful because when designing a general data model for storing mission histories it is important to make an extensive domain analysis to find out the requirements, problems and main topics. This domain analysis is extremely difficult to perform since it is impossible to predict every possible scenario in a DTO. Using this approach, an initial analysis led to an understanding of the problem which could be used as a basis for designing the first draft of the data model. It had to be designed, implemented and tested with different scenarios in mind to exploit weaknesses and limitations of the model.

The initial analysis phase consisted mainly of exploring the existing functionality of MIND since this tool has been used to recreate and explore many DTOs over time. We found several possible solutions that could be used directly in our data model, but the model should also support extensions and other types of data and object which are currently not supported by MIND. Therefore, we also reviewed log files recorded during field trials, documents describing instrumentation plans for field missions

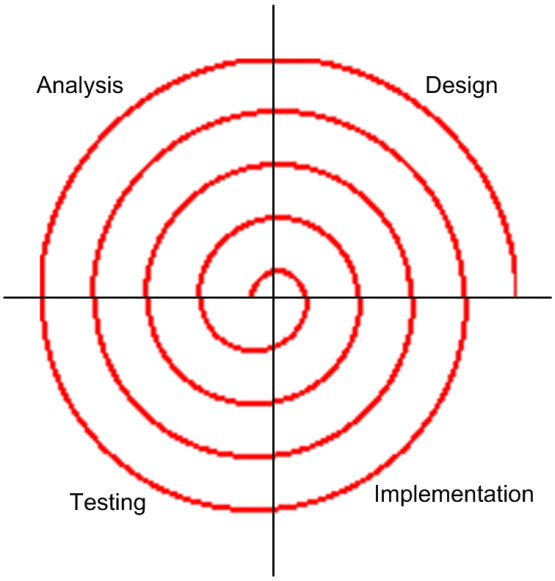


Figure 3 *A simplified spiral model was used as development process when developing the data model for storing mission histories from DTOs, the application framework and the applications.*

and other external documents describing related data models, such as The Land C2 Information Exchange Data Model (NATO, 2002).

During the first design phase we discussed different techniques to solve the problems we found during the initial analysis and drafted an initial diagram of the data model. During the implementation phase the model was implemented in an available database management system (DBMS), SQL Server 2000. The choice of DBMS was not intended to be final, it was simply chosen because a development edition of the system was available at the department. Finally, the model was tested with different scenarios, both in an abstract way, using the diagrams to reason if a certain scenario could be represented using the model, and in a more concrete way inserting records into the database to demonstrate the functionality and confirm the design. As flaws were detected, they were analyzed and the data model redesigned and another step in the iterative process was performed to reach an enhanced model.

1.3.2 Analysis of database management systems

When the data model was considered sufficient we moved on to analyzing what requirements this model put on a database management system and

started to collect data about different alternative systems on the market to find the best suited system for our model.

We had already seen that the model could be implemented in a traditional relational database management system, so the focus was put on finding strengths and weaknesses of other alternatives. The main alternatives to Relational DBMSs that we discussed were Object-Oriented DBMSs, Object Relational DBMSs and specialized systems for temporal and spatio-temporal databases. When we had decided on the type of DBMS best suited we continued to research different implementations of such systems trying to find the best one for our needs.

1.3.3 Development of application framework and applications

The application framework was developed using the same approach as described in section 1.3.1. During the analysis phase we discussed what different applications would be needed and how to model a general framework that could be used to implement this. We also investigated different development platforms and tried to find one that would be easy to work with while still having the opportunity to interoperate with the existing version of MIND.

The analysis led to an initial design that we implemented and tested in C#. In compliance with the spiral model we did it in steps which eventually led to a state where the framework was ready to be tested with real applications. Our initial analysis showed a couple of basic applications that could be implemented to test the most important parts of the framework. The applications where then developed, again using the spiral model, while we continuously enhanced the framework.

1.4 Structure of this Report

In the first part of this report, chapter 1, we describe the background of this project. In the second part, chapters 2 and 3, we continue with the development of the data model for storing mission histories. We further describe the implementation of the application framework in the third part, chapters 4 to 6, while we present the necessary applications for using the database as a storage facility for reconstructing and exploring distributed tactical missions in the fourth part, chapters 7 and 8. In the last part we summarize our conclusions and present our recommendations for future work.

Chapter 2
Data model for storing mission histories

The MIND system has been in use since 1992 for modeling DTOs and has continuously expanded as a research tool. During this process ideas of using a database to store data for recreating missions have evolved. At the start of our project we were presented many features that the MIND research team wanted the database to manage. These ideas were to some extent documented in various reports, but mainly as scribbles and partially formulated solutions. Designing a data model that supported these ideas was therefore an important part of this project.

2.1 Reconstruction and exploration

Reconstructing a DTO generate high demands on the storage structures. The database model designed should be as general as possible to suit the modeling needs of future operations with unpredictable content. On the other hand, some areas require a high level of detail and specialization.

A lot of material is available from previous operations that could be used to get a picture of what we need to store in the data model. For the modeling step of a mission it is obvious that some kind of structure for real-world objects should be stored independent of the studied domain. A suitable way to store the hierarchal dependency of these objects is also important, for example helicopters in a helicopter squadron. The instrument objects for collecting data should also be stored in an easy way, closely related to the objects the instruments is used to collect data for. Structures for storage of data collected by these instruments should also be a vital part of the database model.

As an example the data collection plan of the exercise Cornelia (Morin & Thorstensson, 2000) stated the following data sources to be used:

- Observations – written preformatted protocols
- Communication – recordings of radio transmissions

- GPS positions – converted to preformatted XML files
- Photographs – digital pictures with preformatted protocol attached
- Video – unknown format
- Casualty cards – preformatted protocols
- Police registration - preformatted protocols
- Fax reports – Copies of situation reports
- Tactical information tables – digital pictures

The database model should support extensive analysis of the mission reconstructed and stored. Apart from presenting data and structures from different missions, i.e. easy retrieval of the stored information, it is also vital that new information about data, metadata, can be stored as processed during analysis of the mission.

2.2 Modeling the database

Taking into account the above demands we identified four different main functions and concepts of the system that the data model should support:

1. Setup and store information about units, people, vehicles, instruments and other possible objects that would take part in an exercise or mission. It should be possible to insert and extract the hierarchal dependencies between all parts that have been setup in the exercise/mission. The result stored in the database is a model of the real world mission. This model is normally setup before the mission has taken place.

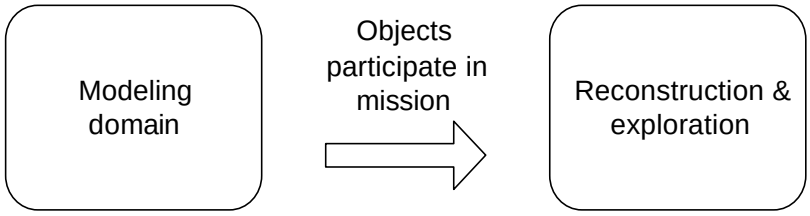


Figure 4 *Top level abstraction of the data model.*

2. Insert data collected by different instruments, for example positions from GPS receivers, audio recordings from radio networks or written observations from units. Today these data are collected during the mission and managed afterwards. The system should be able to automatically insert data provided in a specified format and in the future support real time storage when connected to the instruments.

3. After a mission or exercise, exploration of collected data is an important step for analysis. The database and support systems should allow many manipulations of its data. Examples include adding metadata and classifications to collected data and revising certain parts of video clips and audio files that contain nothing but noise.

4. The system should allow export to several kinds of presentation tools, including the MIND system.

Based on these concepts the model was designed in Microsoft Visio, a tool offering a good platform to build and present database relational diagrams. The model is deliberately built in general terms to allow future changes when requirements on the system are added or changed.

Because of its complexity, the relational diagram of the model is abstracted into blocks, where each block represents a logical part of the databases relational diagram. These blocks are described below. In an additional abstraction these model blocks are divided into two main blocks, see figure 4. These two blocks are separated by the time factor that is critical in this representation. The first, the modeling domain block, represents structures that do not have a time relation. That is, the objects are not associated to any particular event in time where they were used. The second, the reconstruction and exploration block, represents structures that have a time relation. That is, they are objects that have been in active operation or collected data valid for certain periods of time during a mission. The complete data model can be found in appendix A.

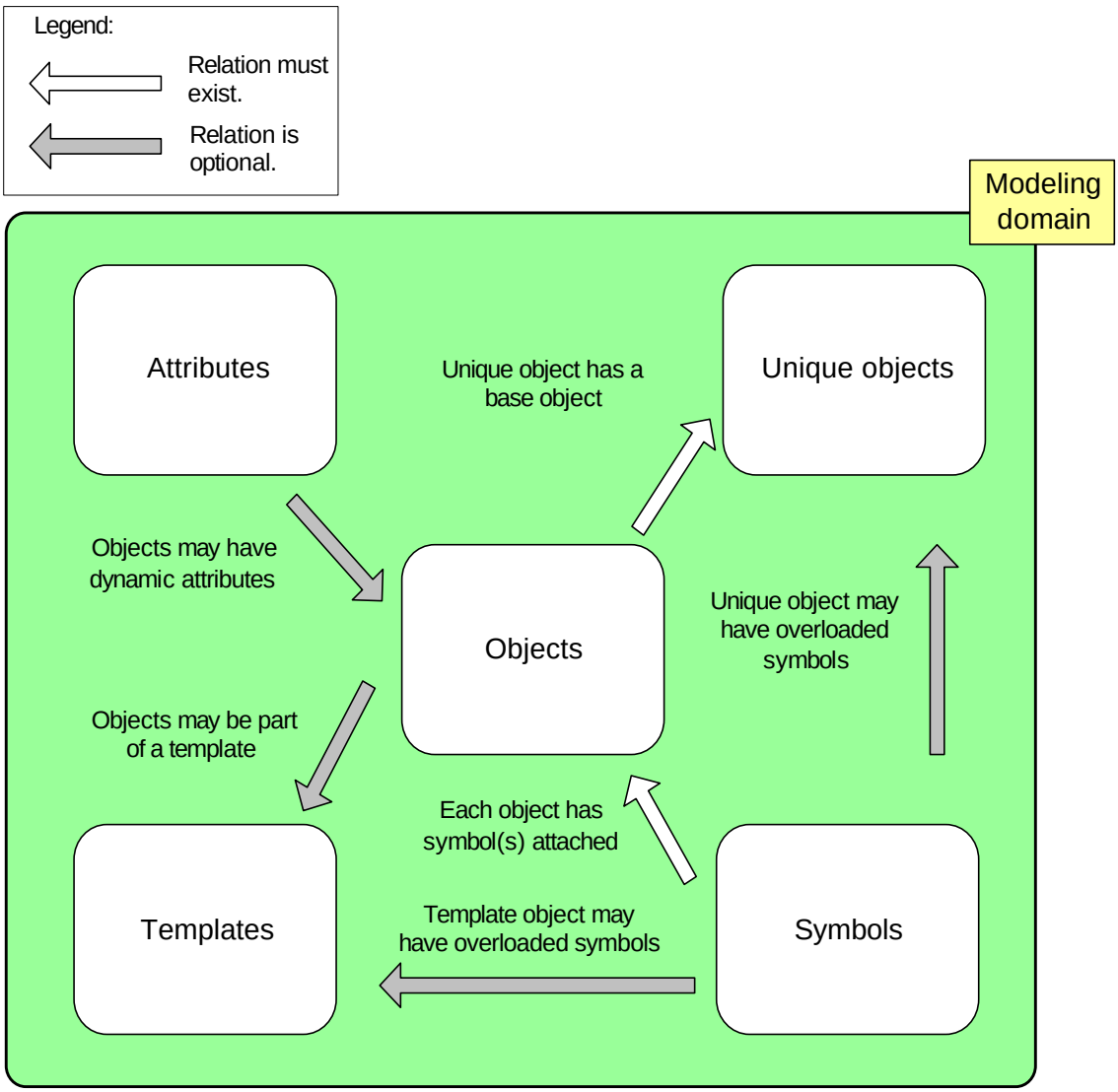


Figure 5 *Modeling domain block.*

2.3 Block: Modeling domain

The modeling domain block holds information about objects and units that the operator creates. This block is divided into five sub blocks with logical differences, see figure 5. These blocks are objects, templates, attributes, symbols and unique objects. Theses parts can be considered to be a representation of the domain with object structures that an operator can choose from when modeling a mission.

The object sub block holds the atomic building parts which are used as components in larger units and composites. The other four sub blocks hold special information or the hierarchal dependencies of these atomic building parts. A description of each sub block is found in the following subchapters including a relational schema of the sub blocks.

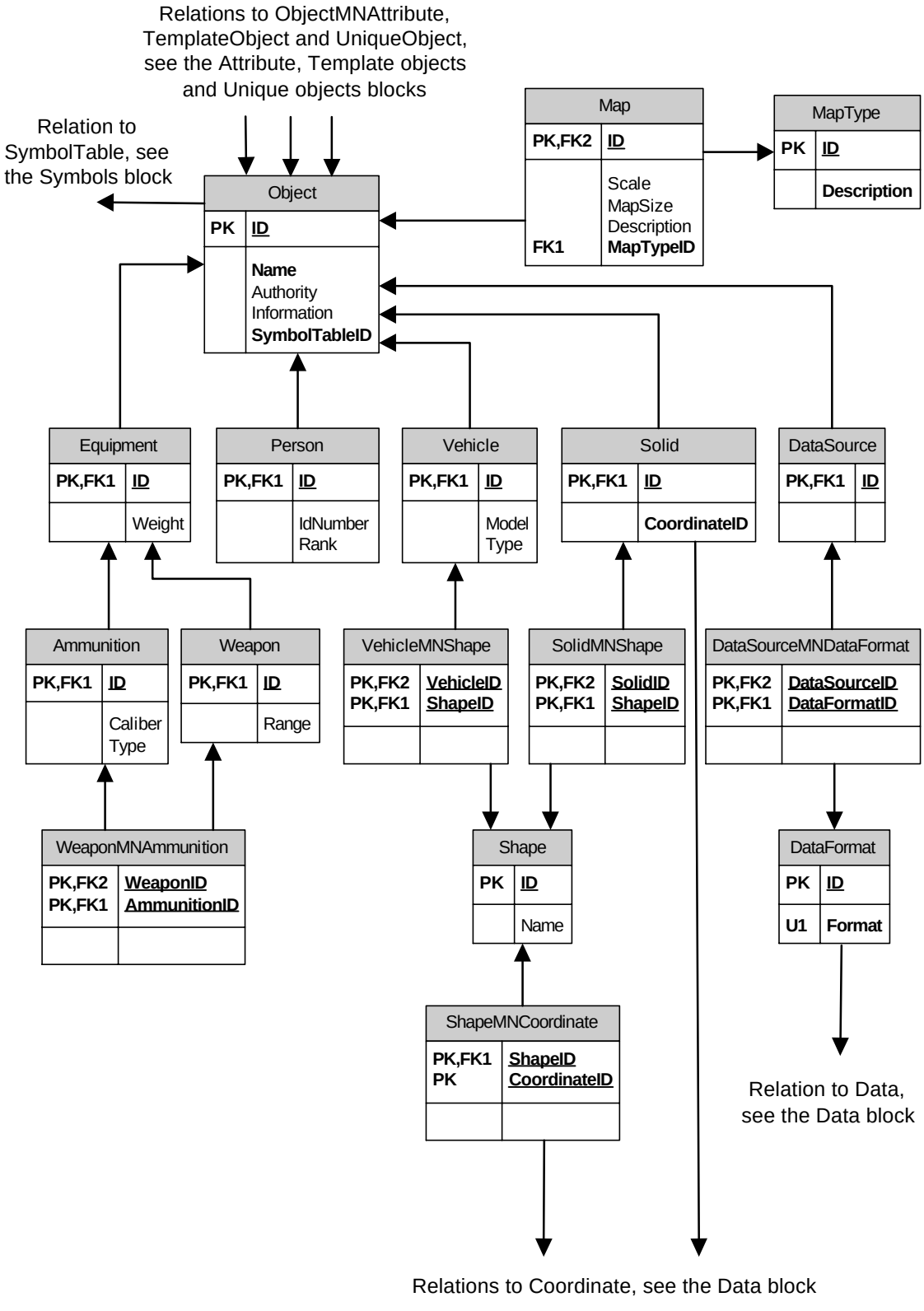


Figure 6 *Relational schema of the Object block.*

2.3.1 Sub block: Objects

In this block all atomic building parts for an exercise or mission are stored in the table Objects. Some information like name and authority are also stored as attributes when the object is created.

For common objects like vehicles, equipment and data sources, they have their own table holding specific information about that type of object. These tables inherit from table Objects, extending it with the special attributes for that kind of object. There are other possibilities to represent different kinds of objects since dynamic attributes can be related to any object. However we identified some classes of objects that would be used commonly and made specific tables and hard coded attributes for them. We believe this would be a more efficient way of handling these commonly used objects, although it is not as general as the dynamic attributes approach. Note that any object, even the specialized ones can have dynamic attributes. In addition, the framework is flexible enough to allow new tables to be added if a new commonly used object type is identified.

The object representation is an abstract view of items. For example an object could be an armored personnel carrier class like ‘Stridsfordon 90’ or a digital camera of ‘Canon IXUS’ type. Therefore several instances of the same object can appear in the same exercise or mission. As an example there can be several ‘Stridsfordon 90’ vehicles on the battlefield or two observers can be equipped with ‘Canon IXUS’ cameras. A relational schema of the Objects block is found in figure 6.

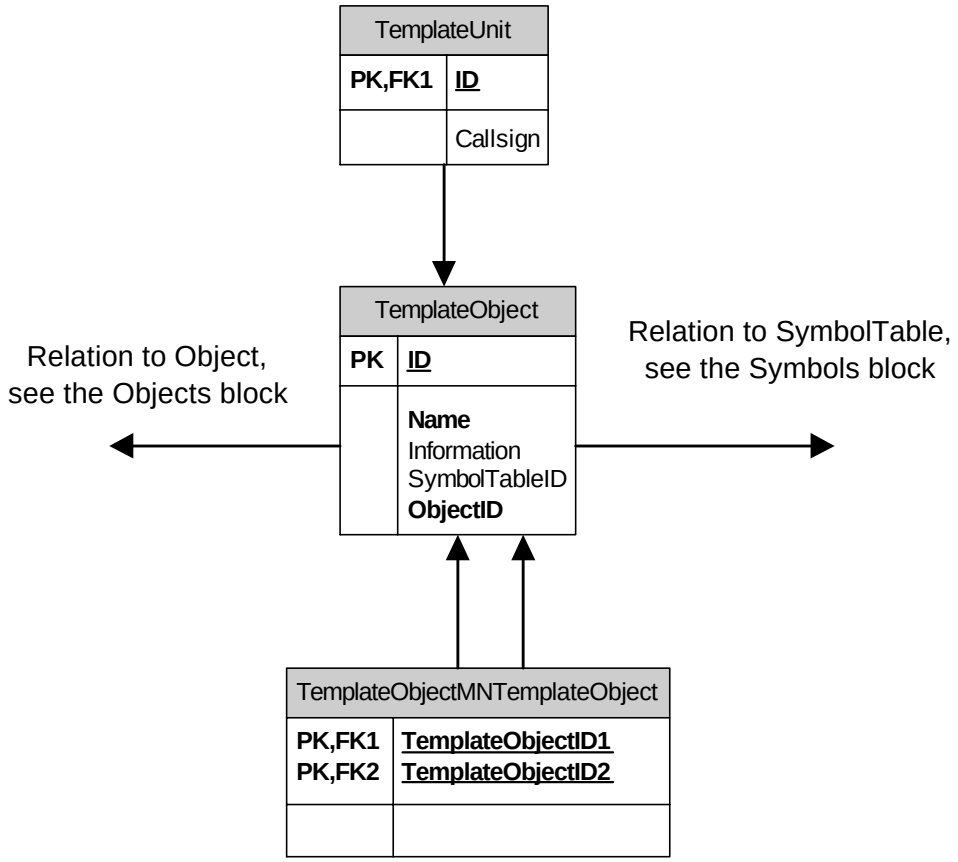


Figure 7 *Relational schema of the Template block.*

2.3.2 Sub block: Templates

The template objects are composite building blocks normally containing several objects. The templates are built as tree structures holding one or many template objects. Each of these template objects has a relation to an object from the Objects table. As an example a helicopter squadron consists of four helicopters type 4. A helicopter 4 is an atomic building block found in the Vehicle table (inherits from Objects). The empty squadron is also an atomic building block found in the Objects table.

Building a template squadron creates a new template object representing the empty helicopter squadron. The template squadron has a relation to the atomic building block squadron found in table Object. To this template squadron it is possible to attach helicopters, or any other type of object, making theses template objects as well. To keep track of which template objects are attached to which, a parent - child relation is also stored. A template object can be appointed a call sign and is then considered to be a template unit. When creating a template object it is also possible to attach a new symbol table. This overrides the symbol table related to the underlying object which makes it possible to have default symbol tables for specific instances of the template.

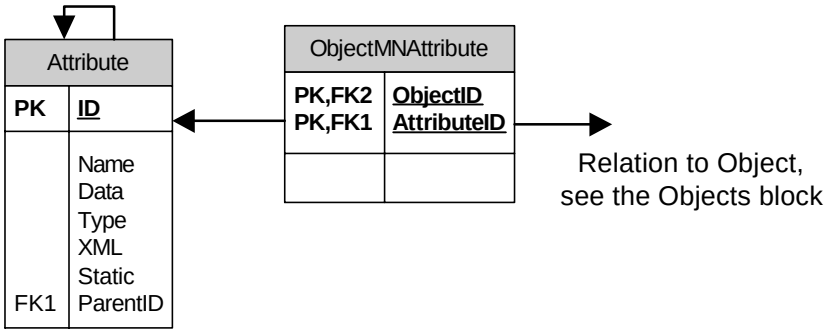


Figure 8 *Relational schema of Attribute block.*

2.3.3 Sub block: Attributes

Attributes can be added to an object to hold vital information. The attributes are stored as tuples <Name, Data, Type, XML, Static, Parent>. A dynamic attribute can thus be considered an object itself consisting of metadata describing the attribute.

The name and data fields are quite obvious; they hold the most vital information of an attribute – what it is called and the data it stores. The Type field stores information about what data type the attribute data has while the XML field can contain just about anything needed to describe the data, for instance maximum and minimum values of the data. The static field is a boolean indicating whether it should be possible to change the data or if it is read only.

The dynamic attributes can be used to store information about objects that are specific, for instance if there is something particular about a specific helicopter, or class of helicopters. Dynamic attributes that are used frequently should be considered to be added to the static model to increase performance.

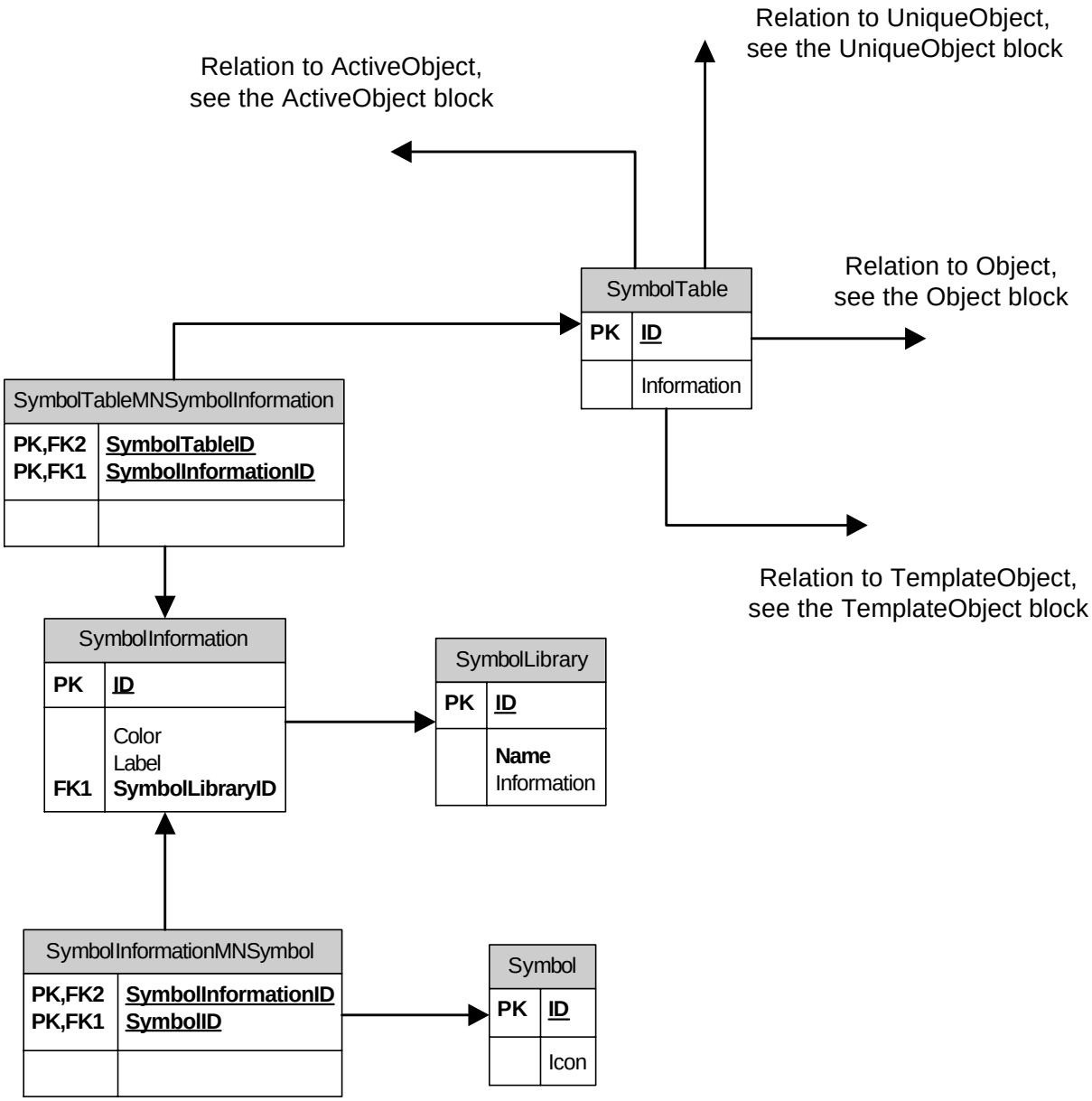


Figure 9 *Relational schema of the Symbols block.*

2.3.4 Sub block: Symbols

The symbol block contains the symbols for different objects. The symbols are graphical items describing the object, often in form of a bitmap or a compressed image. The symbol images are stored as raw data in the Symbols table. Information about the symbol and the color it should be displayed with is stored in the table SymbolInformation.

Each object is related to a symbol table. These symbol tables can be related to several symbol information items. Therefore an object can use many

different symbols and colors depending on the operator’s choice. This is especially useful when dealing with different presentation applications. The application can use several different symbols showing different amount of details, all retrieved directly from the database.

When an object is used in a template object or in a unique object the symbol table relation is inherited. However, it is possible for the new instance to override the default symbol table making the system even more flexible. The same applies when a unique object is used by an active object, the active instance may thus override the unique objects symbol table.

Another important relation is between SymbolInformation and SymbolLibrary. Each symbol can be a part of a symbol library. Using this relation all symbols can be divided into groups. When assigning a symbol to an object it would be convenient to display symbols part of the same symbol library as a possible choice rather than all symbols stored in the database. Examples of symbol libraries are army symbols or naval symbols.

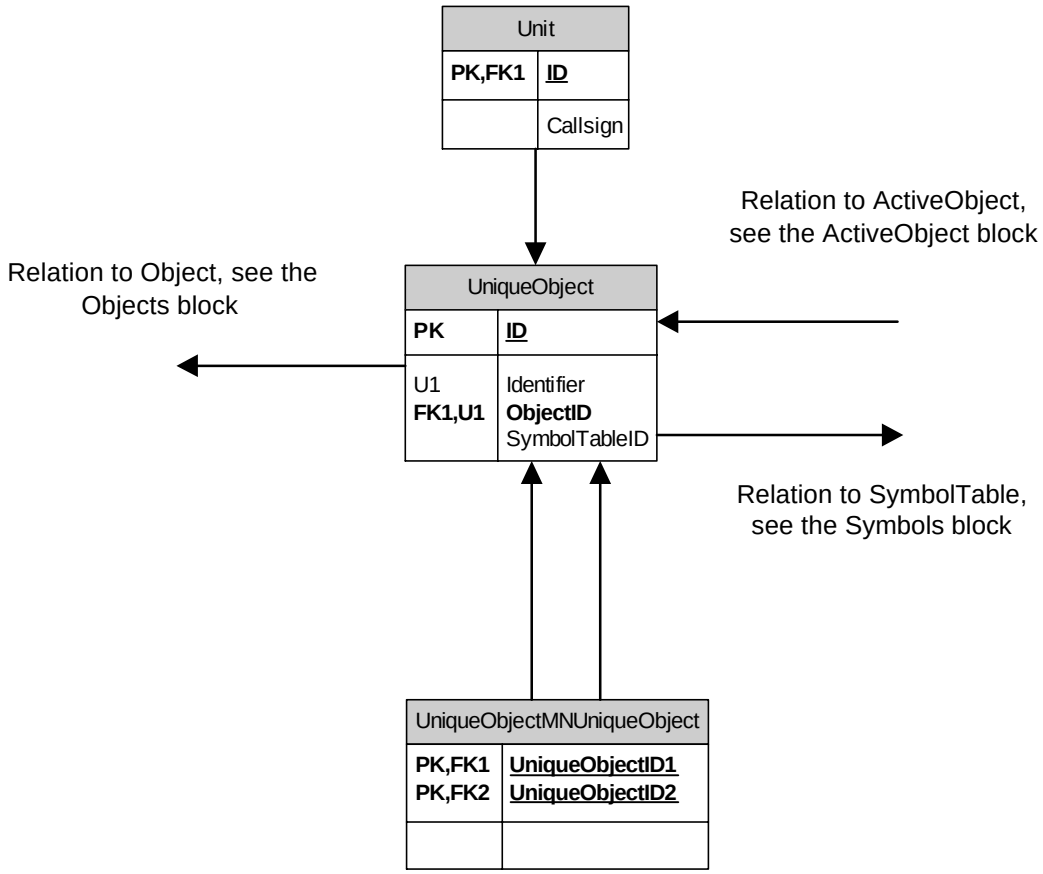


Figure 10 *Relational schema of Unique Objects block.*

2.3.5 Sub block: Unique objects

As the name describes unique objects are instances of objects that are unique. The unique object is identified by a string called identifier. The unique object also has a relation to the Objects table which describes what type the unique object is. As an example, there may exist two instances of the previously described helicopter 4, see chapter 2.3.2, named Y67 and Y70.

While the objects describe types or models in an abstract manner, the unique object is the concrete object that exists in reality. When creating a unique object it is also possible to attach a new symbol table. This overrides the symbol table heritage from the underlying object. Unique objects can, like template objects, also be arranged in a hierarchal structure. Also at this level a parent - child relation is stored to keep track of which unique objects are attached to which.

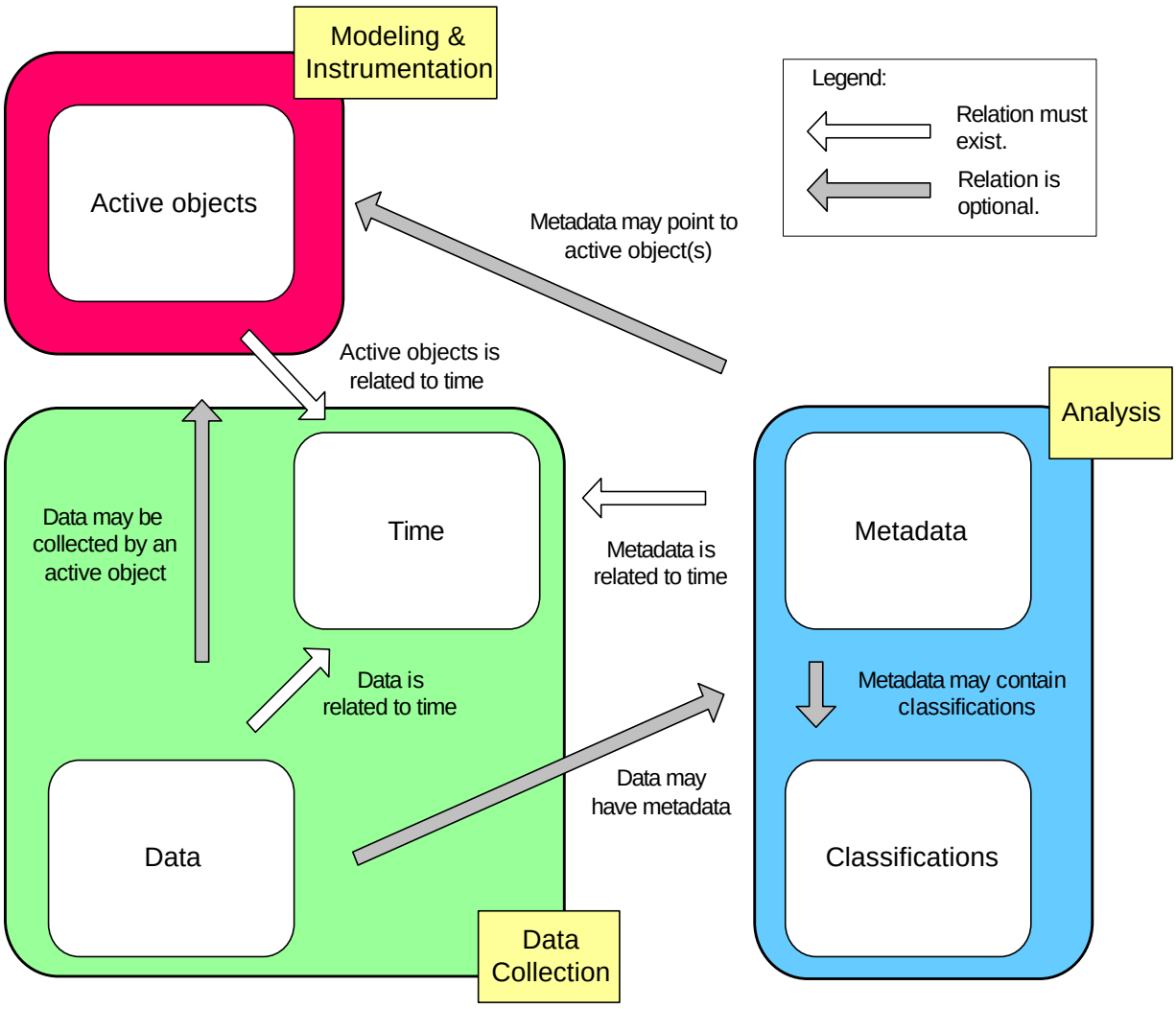


Figure 11 *Reconstruction and exploration block.*

A unique object can also be appointed a call sign and is then considered to be a unit.

2.4 Block: Reconstruction and exploration

The reconstruction and exploration block holds information about missions, objects participating in missions, data collected during the mission and information gathered in post-mission exploration and analysis. The block is divided into three conceptual blocks closely related to the reconstruction-exploration modeling approach (Morin et al., 2000). The modeling and instrumentation block holds the actual model of a mission where participating objects and instruments are stored in the sub block Active objects. The data collection block holds all data collected from the instruments mentioned and stores it in the Data sub block. The time the

collected data is gathered is of extreme importance to be able to reconstruct a mission. Therefore the Time sub block is considered to be a part of the data collection block. The concept ‘presentation’ is considered to be the retrieval of information stored in the data collection and modeling and instrumentation blocks. The analysis block stores comments and notes regarding this presentation.

The main issue for the reconstruction and exploration block is that time is a vital component. Missions take place during time periods, data is produced at certain times in these missions, units are allocated to missions in certain time periods and metadata is created or edited at certain times. A description of each block and its time dependencies and relational schema is found in the following subchapters.

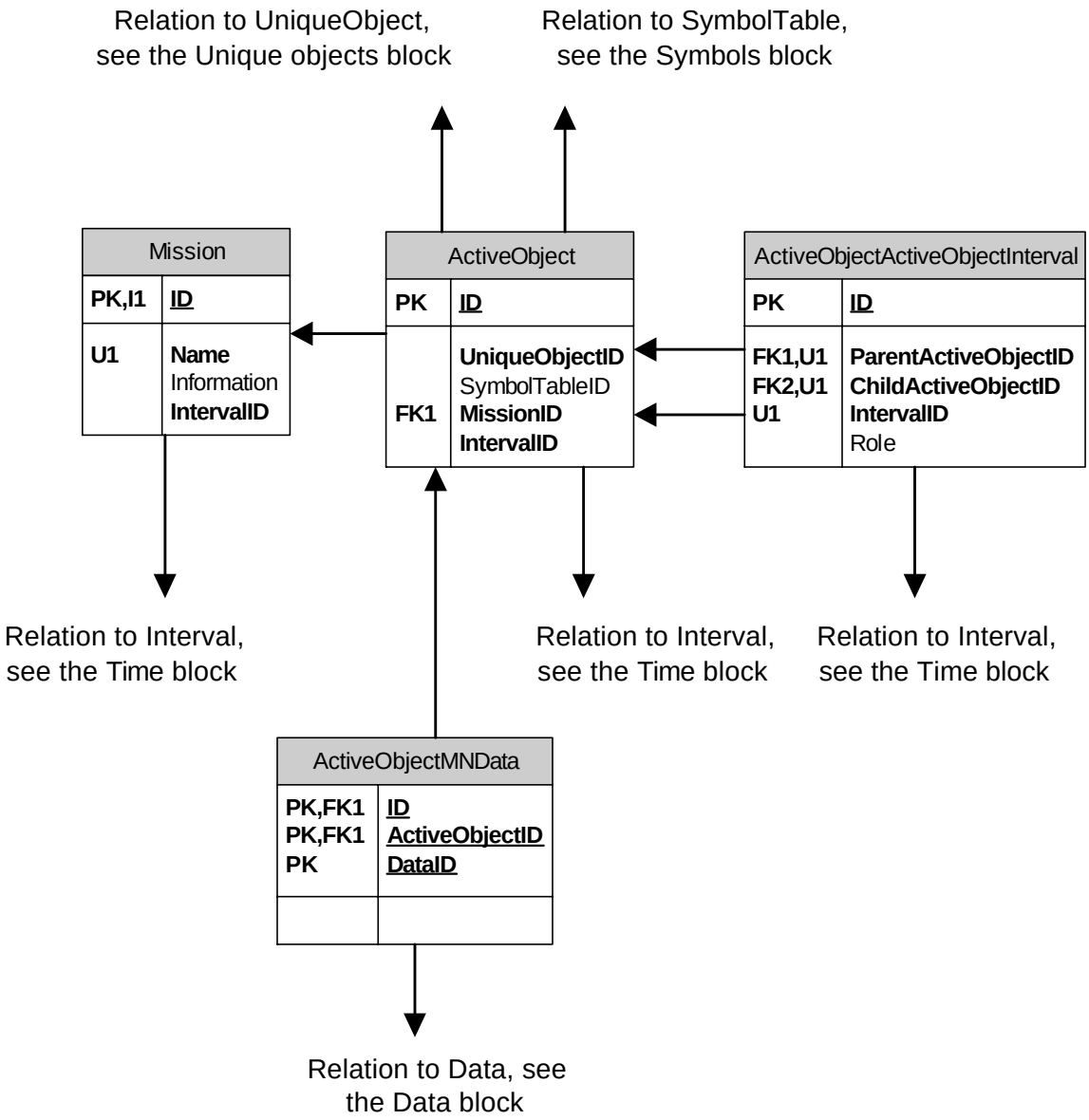


Figure 12 *Relational schema of block Active objects.*

2.4.1 Sub block: Active objects

Tables in this block contain the active objects that take part in different missions. When a mission is created, the time period it is active must also be set. This time period is called an interval, see section 2.4.2.

An active object is a unique object, see chapter 2.3.5, which has been assigned to a mission during a certain amount of time. This amount of time often is the same interval as defined by the mission, but it can be any other type of interval, most likely within the mission interval.

Like template objects and unique objects, see sections 2.3.2 and 2.3.5, the active objects can be assigned to each other in a hierarchical structure with

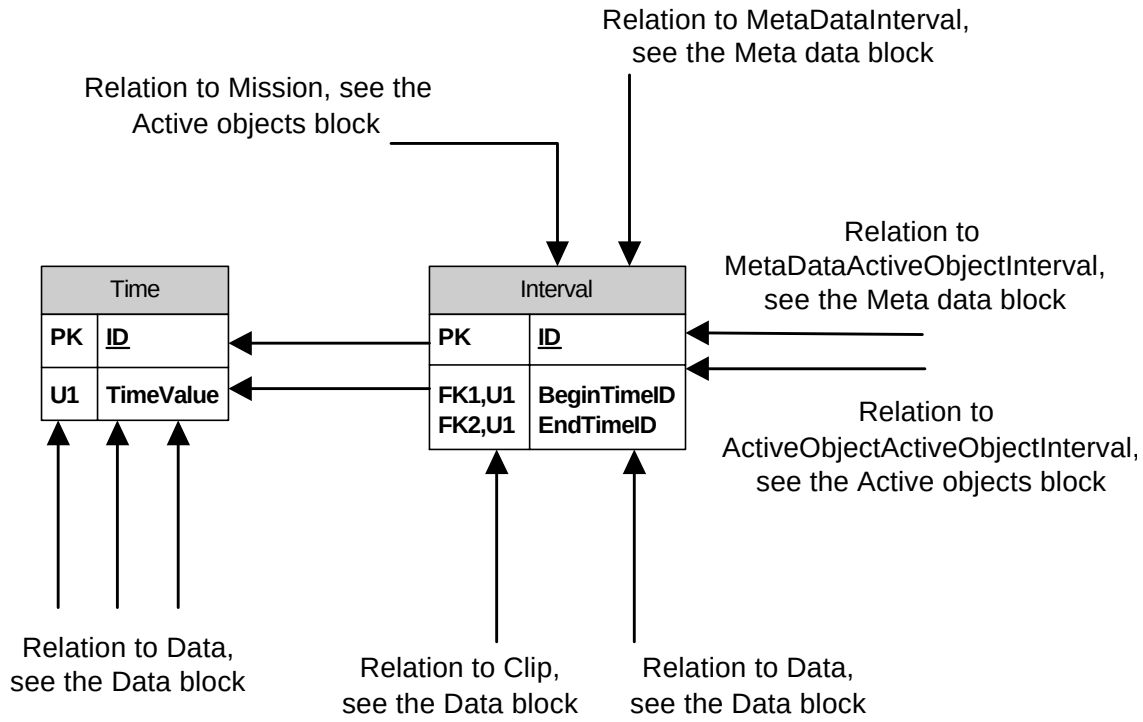


Figure 13 *Relational scheme of block Time.*

parent-child relations. In this relation a role attribute can be set. For example the unique object John Doe can be assigned to an active helicopter as a pilot during a certain time interval. Note that this solution allows the helicopter to switch pilot simply by adding a similar relation during another time interval.

2.4.2 Sub block: Time

The time block is very vital for the functionality of the system. Most events that occur during a mission use a time value in some way. A vehicle position is valid only at a certain time. When analyzing a mission, metadata is produced at certain times and may be edited at another time. There are numerous examples where time should be stored in, or extracted from, the database together with an object. Many tables use a time period instead of a precise time. This is represented in the table Interval with relations to the time table for start and end time.

At an early stage in the project it was apparent that time was not only vital to the project it would also cause some design problems. First of all, what resolution should be used? When dealing with mission periods a resolution of hours would probably be fine. When dealing with helicopter trails the question arise if a resolution of seconds will be good enough. Secondly the

sheer amount of times would cause difficulties. Logging GPS positions every second during a 7 day mission would cause some 600 000 records with time information. Despite the latter we designed the time table as straightforward as possible through a table containing dates and time down to 1 millisecond. This is the default time format used by most database systems.

We are aware of the fact that this may not be the best solution. Performance analysis of different time representations should be done as soon as enough data has been collected and the outcome should be used for a discussion on possible alternatives to our system implementation.

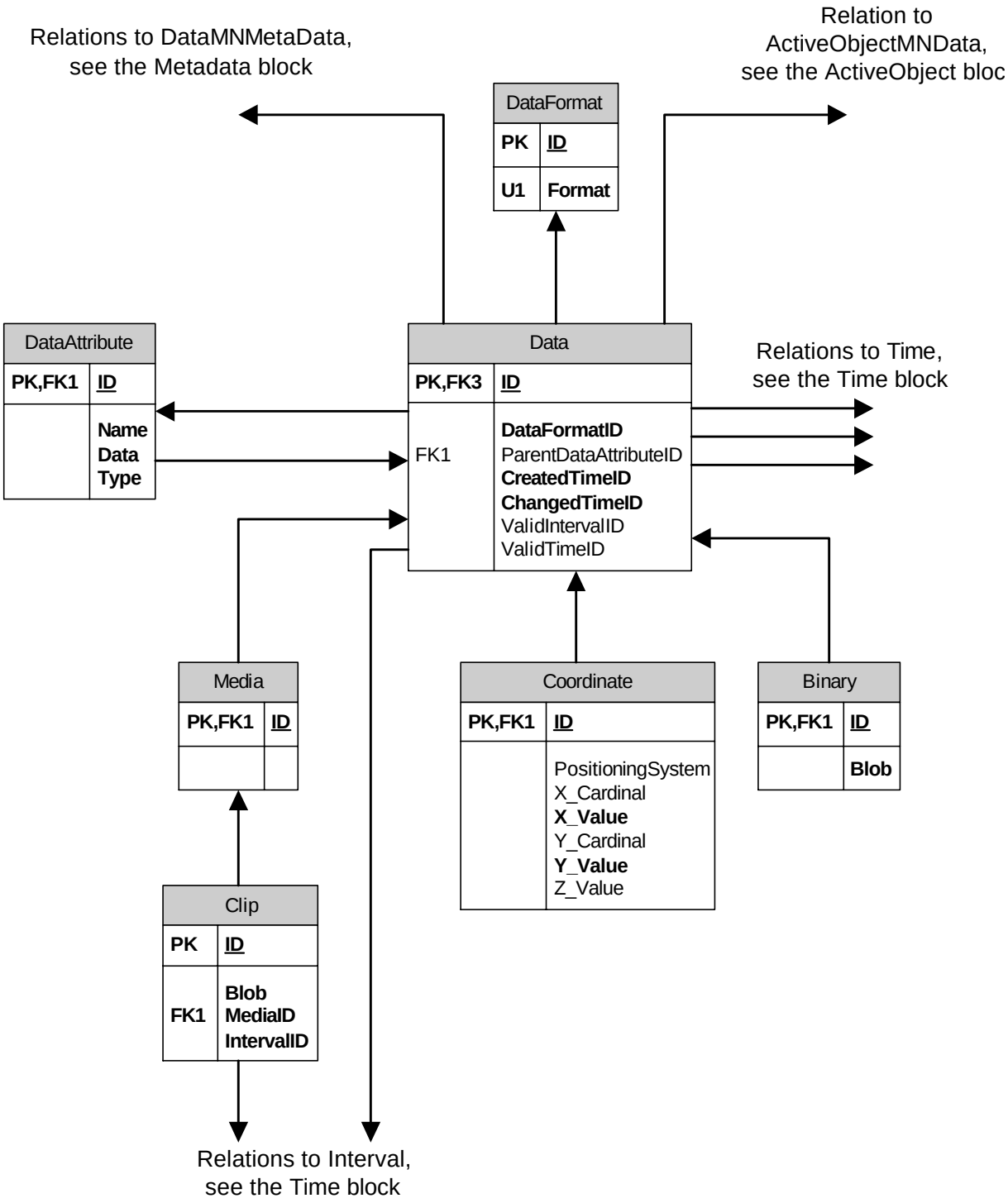


Figure 14 *Relational schema of block Data.*

2.4.3 Sub block: Data

Data collected throughout all exercises and operations are stored in this block. Like the Objects blocks, see section 2.3.1, different classes of data have their own tables inheriting from the main table data. The model

support storing binary large objects, blobs, like audio- and video files. These multimedia files may also be analyzed and split into fragments of the original files. In this way portions of the file that an operator find irrelevant, can be removed and the storage need decreased. Of course this should be used with caution since information that seems irrelevant to one operator can be important to another.

As for objects, the framework allows new tables to be added for modeling future data collection needs, but most types of data should be storable using the aggregate relation to DataAttribute. With this relation a composite data object can be created where every component itself is a data object of some sort. This is usable for instance when storing PIX data, which basically is a composition of a photography and a coordinate (describing where the picture was taken).

Another important feature is the data format table which holds information on how to parse the data. Example may be MPEG-4 which lets the client know that a certain blob is in fact a video stored in the MPEG-4 format.

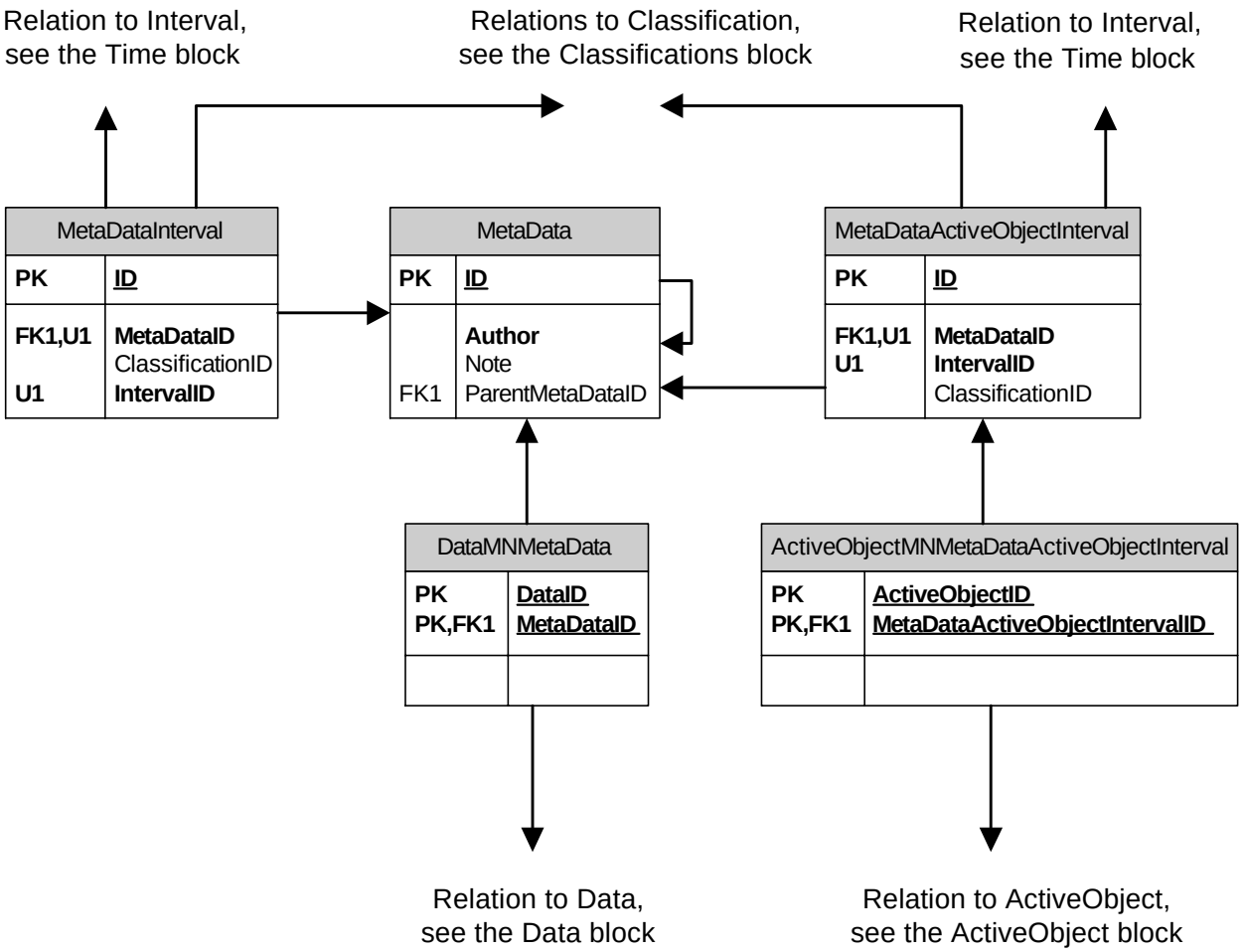


Figure 15 *Relational schema of block Metadata.*

2.4.4 Sub block: Metadata

Meta data is information about other data. In this model it may for instance be notes from an analysis of a sample of data. As an example, information about who is the transmitter on a recorded radio network can be entered as metadata by an operator. Our model supports both simple textual notes and relations to other database entities via the metadata table. One example of an analyzing tool that produces metadata is the Metadata Workbench (Albinsson, Morin & Thorstensson, 2004). The metadata is modeled in MIND, using this tool. However, since the Metadata Workbench has no interface to the database, we have not been able to test our model using it.

With this model data can be collected by an active object, then analyzed and metadata that point to another active object can be created. In the example above this can be used to relate the transmitter, who is regarded as an active object, to the metadata record. It is also allowed to store metadata

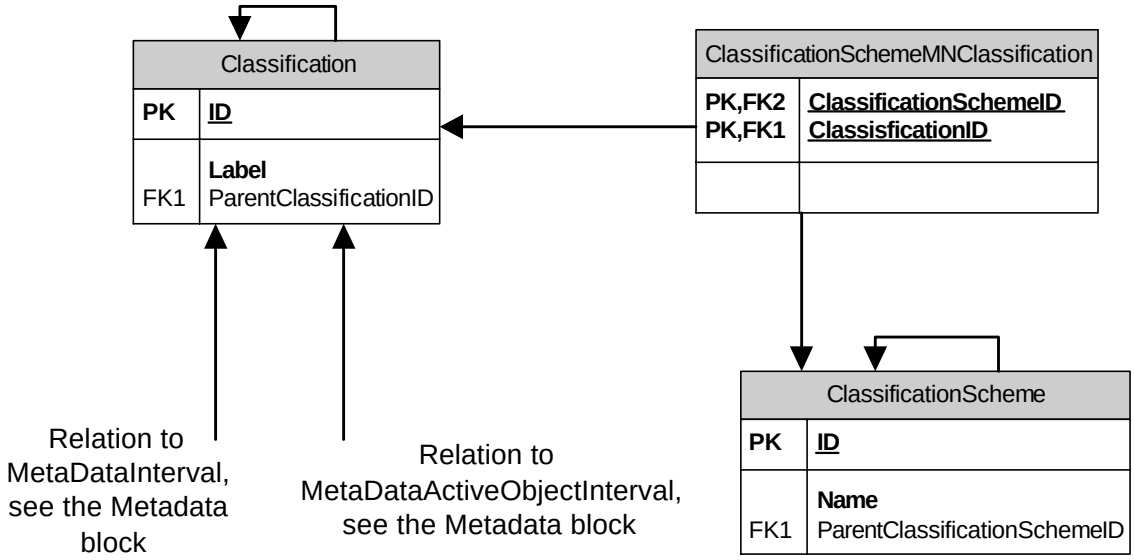


Figure 16 *Relational schema of block Classifications.*

on the metadata if, for instance, an operator disagrees with the previous operator or has more input related to a previous note.

2.4.5 Sub block: Classifications

The classifications block is only used by, and related to, the metadata block. An operator may classify data in a certain way, for instance an operator listening to an audio recording may decide that this audio recording can be classified as an order. Another classification example would be to grade the quality of the sound on a scale from one to five.

To simplify the work for the operator there are also classification schemes which organize the classifications even further. There may for instance be a certain scheme consisting of ten particular classifications that may be used when analyzing video recordings. Organizing these into a predefined scheme will simplify the work for the operator. Note also that classifications may be organized hierarchically, supporting both detailed classification as well as summary classifications (Albinsson & Fransson 2001).

Chapter 3
Selection of database management system

As part of this thesis project we have conducted an analysis of some common database management systems (DBMS) that exist today. In the analysis we have examined both commercial DBMSs and their open source counterparts. We have also examined differences between relational database management systems (RDBMS) and object oriented database management systems (ODBMS). Our conclusions about the different types are described in the corresponding sections below.

3.1 Relational Database Management Systems

RDBMSs are very common these days. There are many well known commercial systems like Microsoft SQL Server[1]. There are also some very interesting open source systems like Firebird[2] and MySQL[3].

The RDBMSs are characterized by the Entity-Relation concept. The database consists of tables where every row is an entity identified by a primary key. By referencing these primary keys entities can have relationships to each other. This solution is very flexible and gives us a very loose coupling between the different data types.

Another strong feature of the relational approach is that the concept is well tested and has been used for many years. RDBMSs are also well-conformed to the ANSI standards[4].

[1] See URL: http://www.microsoft.com/sql/default.asp

[2] See URL: http://firebird.sourceforge.net/

[3] See URL: http://www.mysql.com/

[4] American National Standards Institute, see URL: http://www.ansi.org

3.2 Object-Oriented Database Management Systems

ODBMSs are designed to work well with object programming languages such as C#, C++, and Java. An ODBMS makes database objects appear as programming language objects in one or more existing programming languages. ODBMSs extend the object programming language with transparently persistent data, concurrency control, data recovery, associative queries and other database capabilities.

The main benefit using an ODBMS with an object oriented application for modelling DTOs would probably be that objects would not require assembly or disassembly during execution time. The main problem with the ODBMS is that the standards are not considered as stable as the RDMBS and are therefore more likely to change. This is a problem because a modification to the standards can make old data incompatible with newer versions of the system. It should be noted though that ODBMSs are catching up in terms of maturity compared to RDBMS and vendors are stating that their ODBMSs are operating at many times the speed of traditional RDBMSs.

According to Elmasri and Navathe (Elmasri & Navathe, 2000) mapping binary relationships is not straightforward in object oriented databases, since the designer must choose in which direction the attributes should be included. If they are included in both directions redundancy in storage will exist and may lead to inconsistent data.

Another disadvantage of the ODBMS is that the operations for each object is included in the model and must thus be included in the design in an early stage. With an RDBMS this is not as critical and the operations can be added on at any stage. (Elmasri & Navathe, 2000)

3.3 Object-Relational Database Management Systems

An Object-Relational Database Management System (ORDBMS) is in fact an ordinary RDBMS with an object oriented front-end. Data is accessed as though they were stored as objects but the system implicitly converts data to and from RDBMS format. The consequence of using this form of DBMS is that the programmer needs to produce less code to get a system up and running, but performance will be degraded compared to a conventional RDBMS because of the extra conversion steps.

Some well-known ORDBMSs on the market are the commercial Oracle[5] and the open source alternative PostgreSQL[6].

3.4 Selecting conceptual database management system

Due to the time constraints of this project we had to quickly decide on which concept to use for this model. Our choice was to go for the relational databases (or object-relational ones) because we knew what to expect from them, they have stable standards and they are well documented. Another reason for not choosing an ODBMS is that we need to access binary relationships in both directions, which can be complicated in an object oriented database.

Our analysis showed that even though performance is not as good as with object oriented databases it should be enough since the database is mainly to be used as a component in after-action visualization tools like MIND and thus the time constraints are not that critical. The initial analysis showed that loose couplings between objects is a necessity in order to create a model general enough to cover all requirements the MIND research team has and to allow further extensions and modifications. Entities and relations are well suited to implement this kind of loose couplings.

3.5 Requirements on the database management system

After we decided to use the well known relational systems we had to decide which particular database engine to use. Parameters that we weighed in the decision are costs, performance and programming environment.

When we conducted the comparison we identified some attributes of the engine that we considered vital to our project:

- **Supported platforms**. Even if the current development platform is Windows, there is no reason to stick to Windows when choosing database engine. Since we use a client-server model, we should be able to use different platforms for the client and the server.

---

[5] See URL: http://www.oracle.com/database/

[6] See URL: http://www.postgresql.org/

- **Maximum size of database**. Since the database will contain a lot of multimedia data it is important that the DBMS is capable of handling very large datasets.

- **Maximum BLOB size**. The DBMS must be able to handle very large BLOBs (Binary Large Objects) since the dataset is likely to include many large video and audio clips.

- **Supported DateTime datatype**. After discussions with the MIND research group we found that in the future at least a precision of 1/100 of a second may be necessary when dealing with time in different operations. The DBMS must thus be able to handle time with a precision of 1/100 of a second or better.

- **Supported languages**. Since we want to build applications independent of the underlying database it is important that the database engine supports a standardized variant of the SQL language, like ANSI SQL-92.

- **Application connection**. Since we wanted to use a client-server approach connecting to the database it is vital that the DBMS and our application development environment support a common connection interface.

Database engine	supported LanguageS	Application connection
Firebird	SQL-92 Entry	ODBC, JDBC, C/C++, PHP
SQL Server 2000	T-SQL, SQL-92 Entry	ODBC, OLEDB
DB2 v8.1	DB2 SQL, SQL-92 Entry	JDBC, SQLJ, J2EE, ODBC, XML, OLEDB
Oracle v9i	PL-SQL, SQL-92 Entry	ODBC, JDBC, PHP, ORAPERL, XML, many more
Postgre SQL	SQL-92 Intermediate	ODBC, JDBC,C/C++, Embedded SQL (in C), Tcl/Tk, Perl, Python, PHP
MySQL v4.1	MySQL, SQL-92 Entry	ODBC, JDBC, C/C++, OLEDB, Delphi, Perl, Python, PHP
Jet (Access)	Jet SQL	ODBC, OLEDB

Table 2 *Supported languages and connection types for different RDBMS.*

Database engine	supported Platforms	Database size	Blob size	Time Support
Firebird	Linux, Unix, Windows	32 TB	32 GB	1/100 s
SQL Server 2000	Windows servers	1,048,516 TB	2 GB	1/1000 s
DB2 v8.1	Linux, Unix, Windows	N/A	2 GB	1/1000000 s
Oracle v9i	Linux, Unix, Windows	N/A	4 GB	1/1000000 s
Postgre SQL	Linux, Unix	N/A	Ext store	1/1000000 s
MySQL v4.1	Linux, Windows, Unix	OS dependent	4 GB	1 s
Jet (Access)	Windows	2 GB + links	1 GB	1 s

Table 3 *Storage capacities of different RDBMS.*

In the tables above we have recorded some desired information for each database engine we have examined. The information is collected mainly from Internet resources (Bohuszewicz et al., 2003; Fermi National

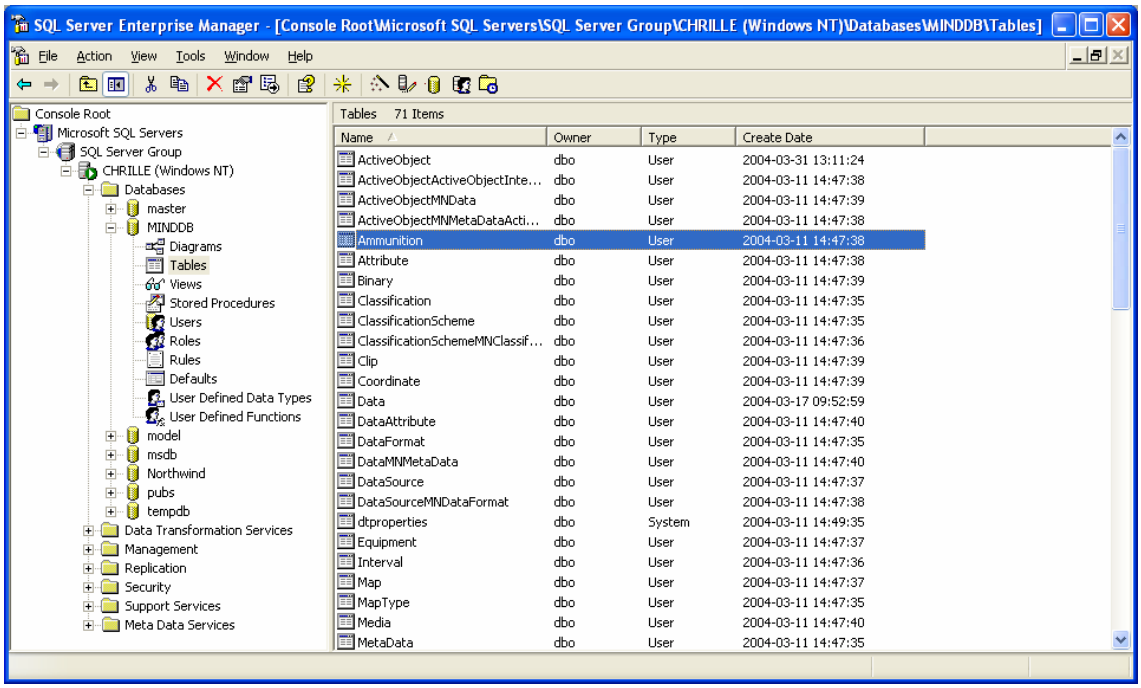


Figure 17 *Designing the MIND database in SQL Server Enterprise manager.*

Accelerator Laboratory/Computing Division, 2003; Microsoft Inc, 2004; Chigrik & Vartanyan, 2004).

Note that the Jet database engine used in Microsoft Access is included in the comparison. We included this DBMS because it is a very common tool that many readers may have experience of. However we excluded it from our list of alternatives in an early stage since the Jet engine lacks client-server model capabilities and possibilities to store large amounts of data.

From the tables we can conclude that most RDBMSs could be used in our project, with the exception of Jet which is limited to 2GB databases, which certainly is not enough for this project. Most DBMSs of today are very powerful if setup correctly. Almost all of the systems can store very large amounts of data and some do not have a limit at all. There are several corporate examples of Terabyte sized databases using the systems described above. Therefore we believe it is more vital that the developer of the system is accustomed to the DBMS to fully exploit the potential of the database engine than decide to use a DBMS based on technical differences. Our intention is also to design the project in such a way that a more thorough investigation of different DBMSs and their effect on performance is possible with a minimal amount of extra work.

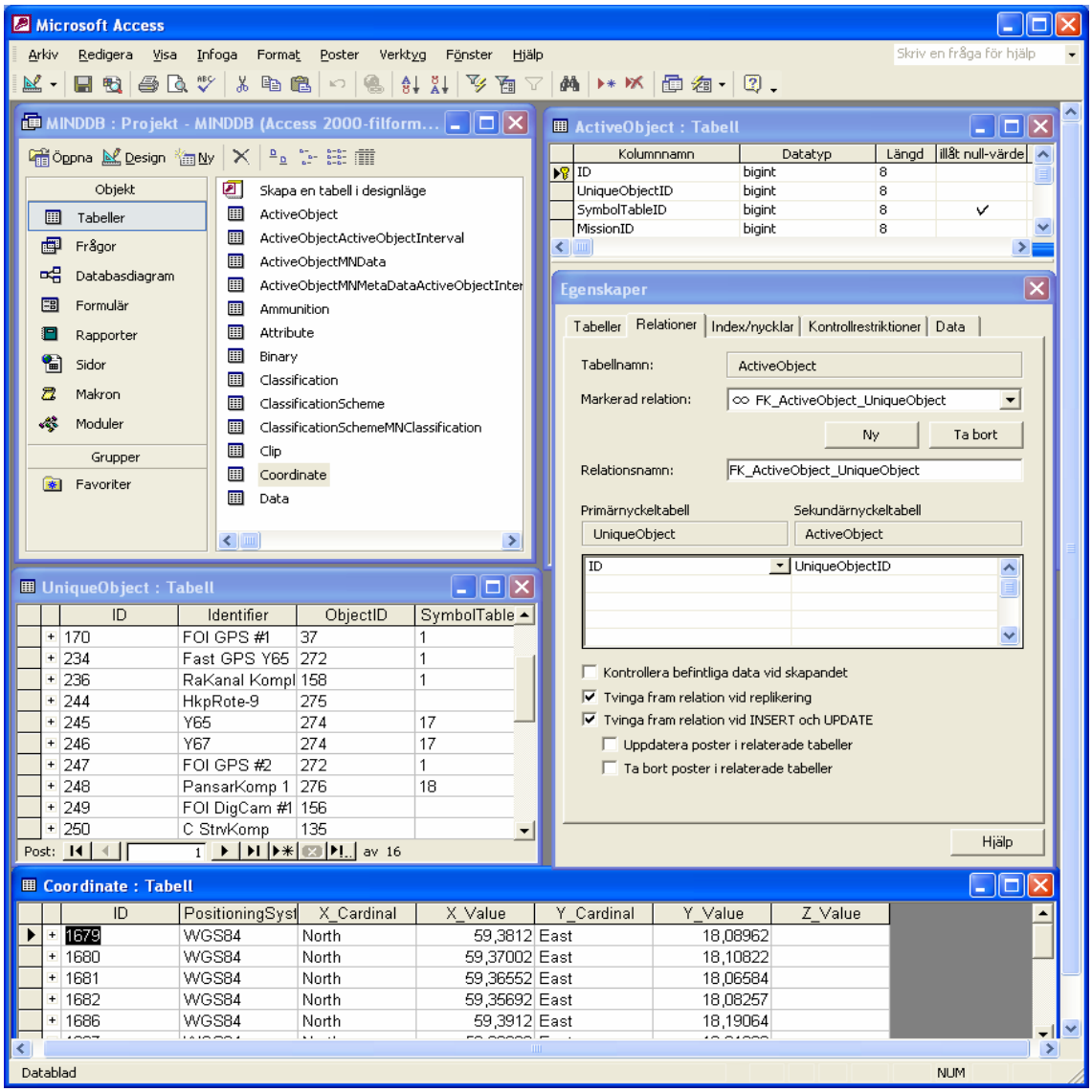


Figure 18 *Manipulating the database via MS Access through an ODBC connection to the SQL Server 2000 database.*

3.6 Implementation of the database

3.6.1 Platform

Facing the fact that the MIND system is developed for the Windows platform, we argued that we initially should choose an engine that supported this operating system. In the developing stage we also prioritized cheap solutions since we did not expect the choice of DBMS to be final.

Comparing performance and storage capacities we found that all of these systems should be able to handle the workload. Our choice was influenced by which products were locally available at our department, which ones

were easy to work with and which ones we had most experience of. Since we also stated in an early stage in the project that it should be easy to change database engine in the future, we felt that our choice was not crucial at this time.

Finally we decided to build our system in Microsoft SQL Server 2000 (development edition) since it was available to us via a department license. It also had an integrated working environment which allowed us to quickly get started.

3.6.2 Implementation tools

We implemented the database strictly according to the data model described in chapter 2.2. This was performed in one of the SQL Server tools, the Enterprise manager, which is a graphical tool that allows the user to quickly set up a database.

The drawback of using this tool is that we are reliant upon Microsoft export routines for exporting the database when wanting to recreate it in another database engine. Another solution would have been to create a script file with SQL commands that generates the database schema when run.

3.6.3 Design choices

The following design choices were made during the implementation:

- All tables, attributes and relations should be created with names according to the data storage model. This makes it easy to relate the database structure to the model.

- Many-to-many (MN) relations should be implemented as separate tables using the name convention <first table>MN<second table>. The MN table primary keys should consist of the two foreign keys from the tables involved. Using standardized conventions makes it easier to program general interfaces and applications.

- All non MN tables should have auto number primary keys called ID. Using primary keys logically separated from the data should be stored eliminates the possibility of editing the primary key.

- Null should not be allowed unless explicitly necessary. This reduces the need to allocate storage space and easier to deal with entities from a programmatic point of view.

- Unique constraints should be added wherever applicable. This to avoid logical conflicts.

- Heritage should be modeled as EER specializations (Elmasri & Navathe, 2000). The special attributes needed are stored in a child table with a relation to the ordinary parent table.

- Referential integrity rules should be set up for tables with heritage. If the object from the special table is removed, it shall also be removed from the parent table.

Chapter 4
The application framework

To simplify the process of building applications using the data storage facility described in the previous chapters, we designed a framework of modules and components that interoperate with the DBMS. After an initial analysis together with the creators of the MIND system, we defined three main uses of the database: setting up conceptual models and instrumentation plans, importing collected data and exploring stored mission histories. These main use cases where the basis on which we designed the application framework.

The main motivation for building such a framework instead of building specialized applications that interoperate directly with the DBMS is to simplify and encourage application development. Using an object oriented mapping of the relational scheme in the database makes it more natural to work in object oriented languages. Another reason for the development of this framework is to allow reuse of code; we found many functions that theoretically could be used in similar applications operating on the database. With a common framework these could be inserted into separate modules and reused.

4.1 Design goals

The main design goals of the framework were set up during the initial design process. The most important features of the desired framework were extendibility, generality, concurrency control, information hiding, independent layers, simplicity and real time support. Each of these goals is described more thoroughly below.

4.1.1 Extendibility

It is very likely that the database schema will change over time when new needs and potential use cases are discovered. The database application framework must therefore allow the user to modify and extend the database schema.

4.1.2 Generality

It is likely that new applications are discovered that should be able to operate, with or without modifications to the database schema. The framework must therefore be general enough to allow new innovative ways of exploiting the database. Even though MIND connectivity has the highest priority, the framework should not be designed solely for a particular application.

While it is important enough to keep the model general at the top, it should also be general at the bottom. Even though the model was only tested using Microsoft SQL Server 2000, the framework should support other DBMSs as well.

4.1.3 Concurrency control

Multiple clients must be able to work concurrently with the database. The application framework should therefore be able to handle concurrency in a well defined manner.

It is important to remember that one application may consist of several threads, each operating as a client towards the database, therefore the framework must also be thread safe.

4.1.4 Information hiding

From a programmer point of view the database structures need not be visible. By using layers and abstraction the framework strive to give the programmer access to the information without the programmer needing to know where it really exists. This can be achieved if presenting an abstract interface, preferably object oriented, towards the data by which the programmer retrieves and stores data.

4.1.5 Independent layers and modules

The framework should consist of several layers that work more or less independently of each other, communicating via public interfaces. These layers must be easy to replace. Every layer should also consist of one or more modules which package data types and functionality with similar purpose. The purpose of packaging functionality like this is to simplify reuse of certain parts of the framework and to simplify maintenance. A module can easily be replaced in the framework simply by replacing it with another module providing the same interfaces.

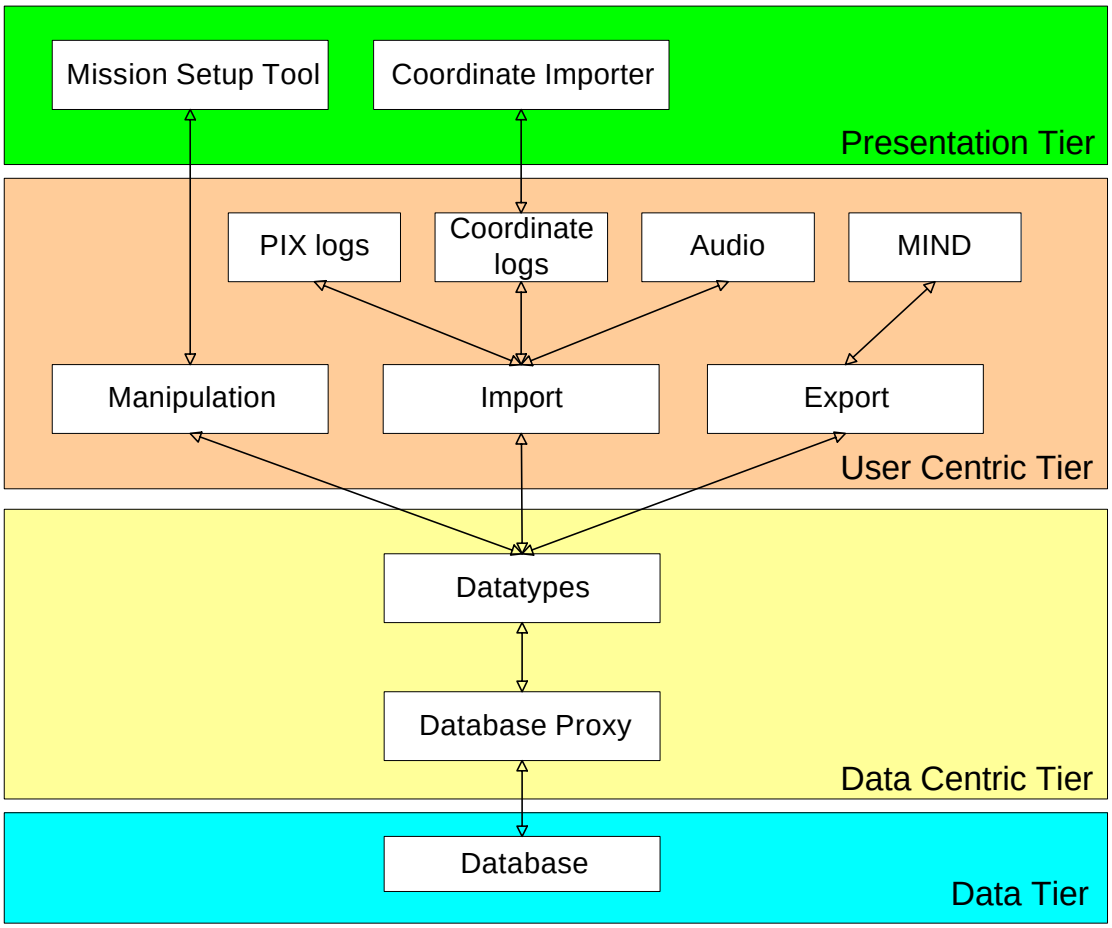


Figure 19 *Layers and modules of the developed application framework.*

4.1.6 Real time support

The possibility to explore DTOs in real time could greatly simplify the work for decision makers. This goal was discussed and has been kept in mind during the entire design process, but it has not actively been designed for since the currently implemented applications are intended for after-action reviews with no priority on real-time support.

4.2 Design model

4.2.1 4-tier architecture

Our model is based on the 4-tier architecture (Moniz, 1999) which is illustrated in figure 19 above. In a 4-tier architecture, all of the data storage and retrieval processes are logically located on a single tier, the Data Tier. In our case this is simply the DBMS and the database.

All requests to and from the DBMS are then routed via the Data Centric Tier which gives the programmer an interface towards the data.

The User Centric Tier is the tier actually requesting data via the Data Centric Tier. It can be seen as a router between the Presentation Tier and the Data Centric Tier. This tier includes the algorithms for determining what data to request and present. The Presentation Tier then presents and receives information from the user.

4.2.2 Modules

Each layer in the 4-tier architecture consists of one or more modules as illustrated in figure 19. Each module is independent in such a way that it can be replaced with another module without interfering with the rest of the modules as long as the new module implements the specified interfaces by which the other modules access it.

The two modules in the Presentation Tier correspond to two different applications developed as part of this project. They are discussed further in the application section, see chapter 7. The modules in the User Centric Tier contain the real algorithms of the framework. The Manipulation module concern data organizing, editing and other manipulation that can be done prior to, or after, collecting data from a mission. It is mainly used to setup the conceptual models and instrumentation plans.

The Import module is used to import collected data into the database while the Export module is used to export data from the database for exploration. The export module is implemented as a COM component which is referenced by a modified version of the MIND framework. MIND then accesses the data in the database via the application framework and can recreate selected samples of the mission histories.

The Data Centric Tier provides an object oriented abstraction of the data in the database. All access to any entities is run via the Database Proxy which handles concurrency and thread safety to avoid database conflicts.

The Data Tier consists solely of the database and the DBMS. The database might be located on a different machine and the server is automatically accessed through the Database Proxy via the network.

A more thorough description of the modules and their implementation can be found in chapter 6.

4.2.3 Extending the model

The model is not supposed to be complete. Extending the model with more modules is a necessity that was kept in mind when designing this framework. The User Centric Tier is the tier in which most work needs to be put. Developing more import and export functions to allow integration with more of the currently used tools for data handling is highly advised. For instance importing video data is something that the Mind framework currently supports, and so should the database framework. Exporting to other formats such as Microsoft PowerPoint and HTML is also a desired extension.

We have identified several more applications that may be developed. Every new application is likely to result in at least one new module in the Presentation Tier. The lower tiers (Data and Data Centric) should not need any extensions. They are fairly static and need to be changed only when the database schema is edited (edit the Datatypes module) or when another data access method is desired (edit the Database Proxy module).

4.2.4 Building applications

Applications are built within the User Centric and the Presentation Tiers. The User Centric modules should contain tools for performing data manipulation, import and export. The applications are then built by tying these tools together via a user interface in the Presentation Tier.

4.2.5 Alternative solutions

The selected model is rather straightforward and suits our needs perfectly. The model was found very powerful and still with much freedom to experiment. Many of the modules can be implemented in several ways. It would even be possible to implement the Data Centric Tier as a service running on the same host as the database. The 4-Tier model above allows this since the only necessary changes would be within the Data Centric module.

We discussed the alternative of implementing stand-alone applications without a framework. We would gain some benefits such as increased performance and the ability to adapt the database mapping as best suited for each application. This approach can still be undertaken if special needs occur for any application.

4.2.6 Maintaining the framework

One important feature of the framework is that it must allow the schema of the database to be altered since the database is primarily to be used for research. It is impossible to foresee all possible applications and needs that could affect the design of the database.

With our object oriented representation of the database with entities as classes this would not be very easy to maintain manually. Every little change in the database might influence a lot of code in the framework. To simplify the maintenance a code generator was constructed. The code generator automatically fetches the schema from the database and generates the core of the framework. After editing the structure of the database the code can thus easily be updated by regenerating the code. Minor modifications to the upper layers might still be necessary if the modifications affected the external interface as well. Still, this should be much less time consuming then revalidating the entire core.

Chapter 5
Implementation techniques

During this project several implementation techniques had to be investigated to find the best solutions to given problems. In this chapter we describe how the system was implemented by presenting the programming environment in which we work and how the DBMS was integrated into the application framework.

5.1 Programming environment

The programming environment mainly used for this project is Microsoft Visual Studio .NET 2003 and Microsoft C# .NET [7].

5.1.1 Platform

When this thesis was written the latest version of the MIND framework that was available is written for Windows using Visual C++ and COM. One of the primary goals with this project was integration with MIND. Therefore we needed a programming environment which was fairly easy to make compatible with COM. Early investigations showed that a native COM or .NET environment was to be preferred.

Since our framework should be used as a platform for many different projects we wanted to focus on creating an easy-to-use API which is easy to extend and work with. The modern .NET framework has a very programmer-friendly API and the structure of it is more natural to most programmers than the COM approach.

5.1.2 Programming language

Any of the languages supporting the .NET framework could have been selected, but we have chosen C# since we found it very well integrated with .NET and easy to work with.

---

[7] For information about Microsoft products and concepts see URL: http://msdn.microsoft.com/

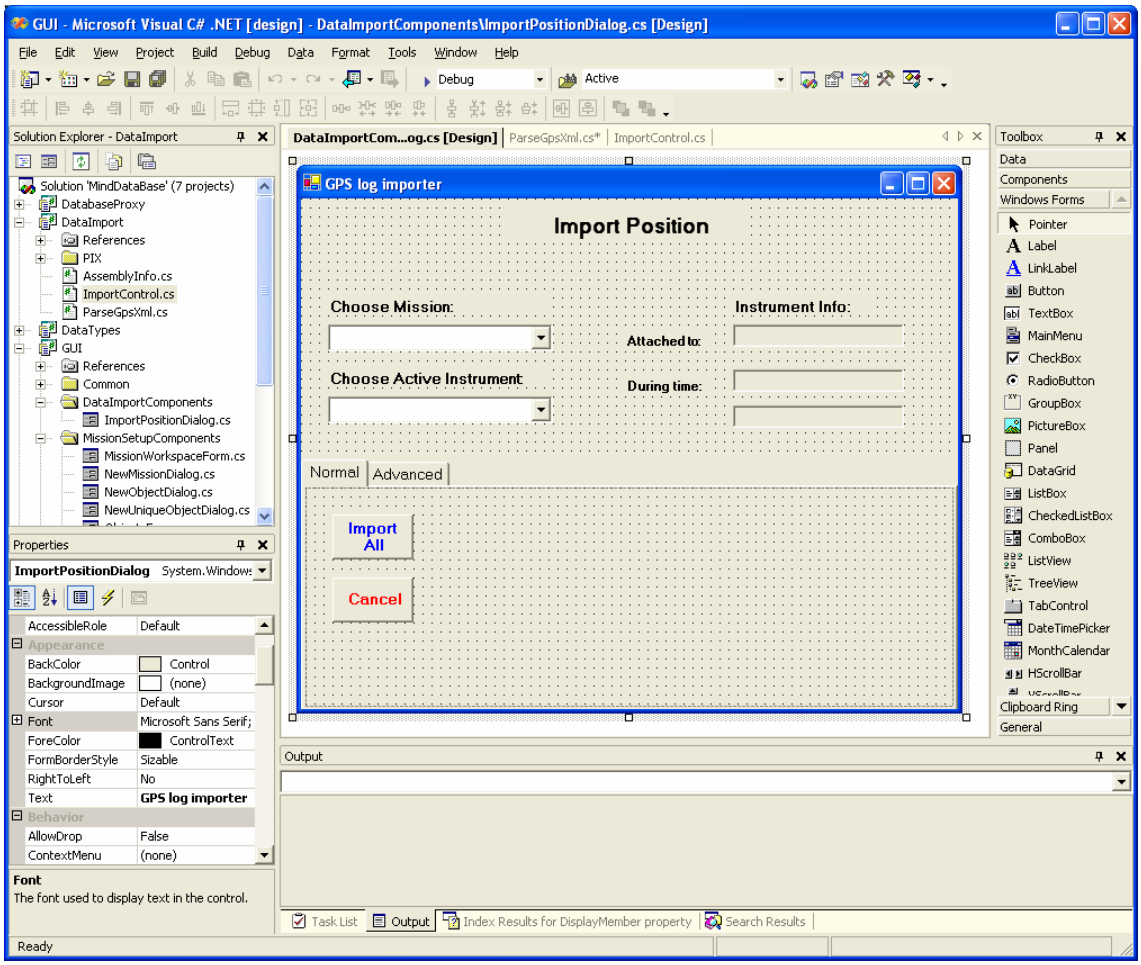


Figure 20 *Using Visual Studio .Net 2003 to design the GPS log importer GUI.*

Some modifications to the MIND framework were made in Visual C++, see chapter 8.

5.1.3 Programming IDE

Using Visual Studio .NET 2003 felt rather natural to us since we had chosen to work with .NET and C#. This IDE has many attractive features to a .NET programmer which essentially simplified our work, see figure 20. For the MIND modifications, we used Visual Studio 6.0.

5.2 Interoperability

When we had decided to use .NET as the main platform for our implementation we needed to solve the issue of interoperating with the MIND framework which is based upon COM. There are two ways for doing this, either by using COM object from .NET or the other way around. We examined both methods to find the best suited solution for our project.

Based on our findings, presented in appendix D, we chose to create COM components using .NET to interconnect the database framework with MIND.

5.3 Database integration

The core of our framework is of course the database. Integration with the database is therefore very important to our task. .NET supports the database connection API OLE DB, which has been used for a long time and therefore can be expected to be thoroughly tested. OLE DB came as a natural choice to us because of its stability and extensive documentation.

There is also an alternative connection method designed by Microsoft to operate with SQL Server. This solution is reported to improve performance but was not selected because it would tie the framework to SQL Server since no other DBMS currently supports it.

5.3.1 Selected solution

OLE DB is operating in a disconnected mode which means that all operations are executed on a local copy of the database and not committed until the user specifically requests a commit. This ensures good performance locally, but the programmer will never be certain that the data he is operating on is still valid unless he uses transactions. This solution does not fit well with our attempt to hide the database and its implementation from the application framework.

One solution was to simulate connected mode by using transactions to retrieve the latest value every time a value is requested from the database. This is also the solution we selected as the first one to try out in this project. The main benefit of the above solution is that the local data will always match the database data when used. Another important benefit is that this solution allows us to create a framework where the database is completely hidden from the programmer.

A drawback is that the suggested solution is very slow. All access to any data in the database requires at least one (sometimes many more) queries to the database. Tests have also showed that this may introduce a heavy load on the network, even on a 100 Mbps Ethernet network between the database and the client, the network proved to be main bottleneck in some applications. Another drawback is that from a single client point of view the data model may seem to be undeterministic since one entity may

change between two instructions if another client modifies data while the first client is doing something else. It is of course possible to lock an entire section of data, but with a complex model like this one it comes with a risk of large chunks of data being locked for a long time and in the end deadlocks are likely to occur.

5.3.2 Suggested improvements

While the model outlined tries to simulate connected mode to always ensure the latest value is retrieved from the database, this might not always be necessary. In some situations we might simply not care whether the data is the current data or the data 10 seconds ago. If this is the case, then it would be enough to work in a disconnected mode. Simply adding an option for the programmer to go disconnected would be an alternative that would improve a lot of applications in terms of speed. This solution would be fairly easy to implement, however with this solution we would be back to were we started, i.e. the programmer needs take care of any concurrency control.

Another model that was suggested was to create a service that is always connected to the database and to which the clients may request subscriptions to the database. When data is updated the service will then notify those clients that are interested in this data. A solution like this would probably make the framework a little more complicated and it would certainly not be as easy to implement as the suggestion above. Still, it is our recommendation that this solution is the way to go when improving the application framework. It should also be noted that the service should be executed on the same server as the database is running on to reduce network load.

Chapter 6
Implementation of the application framework

The application framework is designed as a platform on which applications for reconstruction and exploration of DTOs can be built, see chapter 4. This chapter is a brief specification of the implementation of the framework.

6.1 Requirements

Most requirements on the application framework are in reality requirements on the design. Still, a few of the design goals, such as simplicity, need to be taken into account also for implementation of the framework. Apart from these design goals, the following requirements are set up for the implementation:

6.1.1 Object orientation

The framework is written in an object oriented language to allow desired features such as inheritance and interfaces. The mapping between the relational schema and an object oriented model is straightforward and implemented in accordance with the method described in (Elmasri & Navathe, 2000).

6.1.2 Modularity

The modules identified in the design are implemented as separate modules or projects. Each of the modules should correspond to one library which should be tested separately and regarded as an independent component of the framework. This solution simplifies the process of building additional extensions to the framework as new modules are ready to be plugged in without the need for any modifications to the original framework.

6.1.3 Relocability

It should be easy to distribute new, or replace old, modules in an existing installation of the framework. Allowing the administrators to add modules simply by installing extra content on the previous release will simplify future maintenance as modules are likely to be added or changed.

6.1.4 MIND compatibility

As MIND is the main research tool used for DTOs and command and control analysis, the process of evaluating the application framework is greatly simplified thanks to the possibility to export data from the database to MIND via the framework.

6.1.5 Implementation language

The selected programming language for the application framework is C#. This language is selected due to the fact that it is very nicely adapted to the .NET framework and that it resembles popular languages like Java and C++.

6.2 Data Centric Tier

6.2.1 Database Proxy

The implementation of the database proxy is very central to the entire framework. The proxy provides a fairly easy-to-use interface towards the Data Tier. This class reads a configuration file and extracts data about the database from it. This data is then used to set up an ODBC connection via the .NET OLE DB classes towards the database through which all the communication with the database is sent.

6.2.2 Data types

For each entity in the database there is one data type class. The columns in the tables are mapped to properties in the corresponding classes. Foreign keys and MN relations are mapped to collections of the target data type. A complete description of the mapping schema used can be found in (Elmasri & Navathe, 2000).

Every access to a property of any data type is routed to the database via the Data Tier and the result is then returned to the user. The constructor of each data type is designed to automatically create a row in the corresponding table each time the class is instantiated. This means that all objects are automatically persistent, without the need for any particular user interaction.

6.2.3 Data type generator

The mapping between the entities and the data types is very static and instead of writing every construct manually, we decided to create a data

type generator that does the work for us. The data type generator connects to the database and fetches the schema; it then produces the code needed to generate the DataType module. This approach gives the benefit that we only need to rerun the generator and recompile, after the database schema has been altered, to keep the framework synchronized with the database. Maintenance is thus largely simplified by the use of a data type generator.

6.3 User Centric Tier

6.3.1 Manipulation

The manipulation module contains several functions for manipulating the data in the database in some more or less complex ways. The module currently consists mainly of algorithms that are supposed to be used by the Mission setup tool, but this module is intended to grow as more applications are developed.

6.3.2 Import

The import module contains common functionality for the log file importers, such as a small framework for parsing XML files and structured storages[8]. The module is very small and likely to grow as more data formats are imported into the database.

6.3.3 Coordinate log import

All specific functionality for importing coordinate log files, such as the ones generated from GPS logs (appendix C), is found in the Coordinate log import module. The XML reader framework from the Import module is used to parse the file and the DataType module is used to store the data in the database. The module is an example of a module used for data collection. Many similar modules can easily be built to import other types of collected data, the PIX and Audio log import modules described below are two such examples.

6.3.4 PIX log import

PIX (Morin, Jenvald, Nygren, Axelsson & Thorstensson, 2003) is a tool developed by the MIND research team for creating log files with metadata

[8] See URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/stg/stg/structured_storage_start_page.asp

for digital cameras. The log files include information such as when the picture was taken, who shot it, notes and more, see appendix D. The PIX log import module work in the same way as the Coordinate log import module, i.e. by parsing an XML file and storing the data in the database. The main difference here is the storage format. The coordinate logs are fairly easy to store in our model as there is a specific coordinate table in the schema. PIX data on the other hand, is treated as an aggregate of photographs and coordinates which complicates the storage procedure.

6.3.5 Audio log import

The audio log files are currently stored as structured storages, and thus have to be parsed as such. The Audio log import module therefore uses the structured storage features of the import module to retrieve the sound recordings and then stores the sound as clips in the database.

6.3.6 Export

The export module is currently empty, but it is supposed to contain common functions that are needed for exporting data from the database to external formats.

6.3.7 MIND export

The MIND export module contains a few functions adopted for exporting data to MIND via a COM interface.

6.4 Presentation Tier

Two main tools where developed in the presentation tier, below is a brief description of the modules while the applications themselves are further described in chapter 7.

6.4.1 Coordinate log importer

The coordinate log importer is a simple graphical front-end to the Coordinate log import module found in the Data Centric Tier. More information about the Coordinate log importer can be found in section 7.1.

6.4.2 Mission setup tool

The mission setup tool contains a graphical user interface with which the user may build an organization hierarchy before any mission. The main focus is usability; the GUI should be intuitive and easy to use to simplify

the work for the user. This tool is the main tool used to set up the conceptual model and the instrumentation plan before the data collection phase. More information about the mission setup tool can be found in section 7.2.

6.5 Testing applications

We tested the lower tiers thoroughly after each of the implementation phases in accordance with the spiral approach described in section 1.3. To test the user centric and presentation tiers and verify their functionality, on the other hand, we built specialized applications that used the most important features of these modules.

Apart from the specific tests described below, we have performed several tests to measure performance and verify thread safety for instance. We do not claim to have tested the entire framework, but our tests indicate that it is fairly stable.

6.5.1 Manipulation

We have mainly tested the manipulation module using the mission setup tool described in chapter 7. The tests showed that conceptual models and instrumentation plans can easily be built for several scenarios in the army, naval and rescue operation domains.

6.5.2 Import

The coordinate and PIX log import modules are tested thoroughly to verify the data collection part of the database. Data can successfully be imported to the database and linked to the instruments that collected the data. Apart from these tests, we have written several simple programs to test specific features of the framework, such as image retrieval and storage.

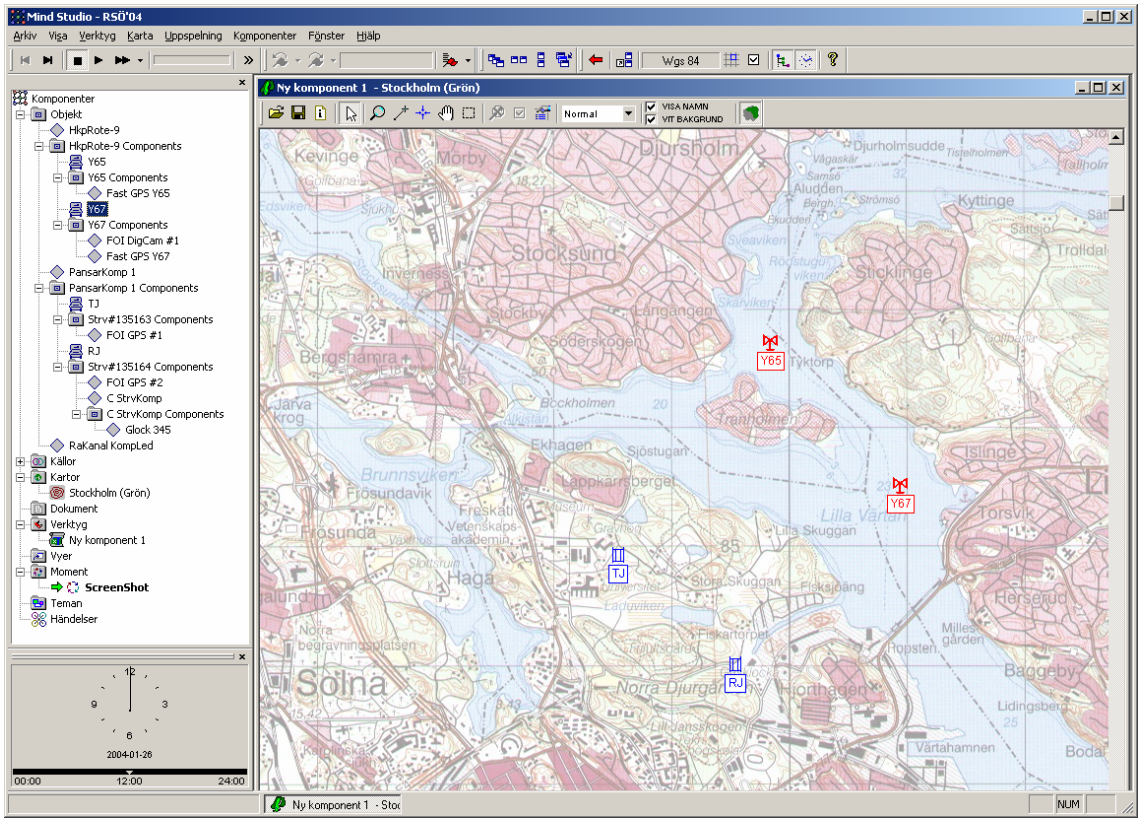


Figure 21 *This scenario was recorded in central Stockholm, Sweden and has been stored in the database. On this picture it has been recreated and is being reviewed in MIND using the application framework to access the database.*

6.5.3 Export

The MIND export module provides an excellent tool to test the entire framework and data model since one of the primary goals is to create a data storage concept which can be used together with MIND. We modified the core of the MIND framework to successfully import the data stored in the database. We have also reconstructed several operations in MIND, including a helicopter wing flying over central Linköping in March 2003 and a fictive mission where a battle tank squadron and a helicopter wing are maneuvering in central Stockholm, see figure 21. The MIND modifications are further described in chapter 8.

Chapter 7
Applications

An analysis together with the MIND research team led to the description of several different applications needed to reconstruct and explore DTOs based on the data model defined in chapter 2. We identify the need for tools for building conceptual models and instrumentation plans, for importing collected data files and for exploring and analyzing the mission history.

Before this project MIND was used for all these steps, with main focus on the exploration step. We still use MIND for this exploration by adding a simple connection to the database. For the reconstruction phase though, we choose to develop new applications since we believe it is currently very complicated to implement the database connection for these tasks in MIND. We are also building simple tools for importing collected data since these tools are easy to create and thus they provide quick verification of certain portions of the system.

Having built these applications and the connection to MIND, described in chapter 8, we have the tools for demonstrating the entire flow from modeling a DTO to the exploration of the recreated mission history.

7.1 GPS log importer

The GPS log importer is the first tool that we developed for operation with the database. The GPS logs were selected because of their straightforward structure and because of the large amount of GPS logs available at the Swedish Defense Research Agency.

Originally the GPS log importer was used to test whether the parts related to storing and retrieving data and coordinate entities are useable, but the importer is also a much needed tool in itself. Almost every data is somehow related to a position, and many observers or participants of a mission are equipped with GPSs, so we can assume that large quantities of coordinates are going to be stored in the database and tools like this one will be frequently used.

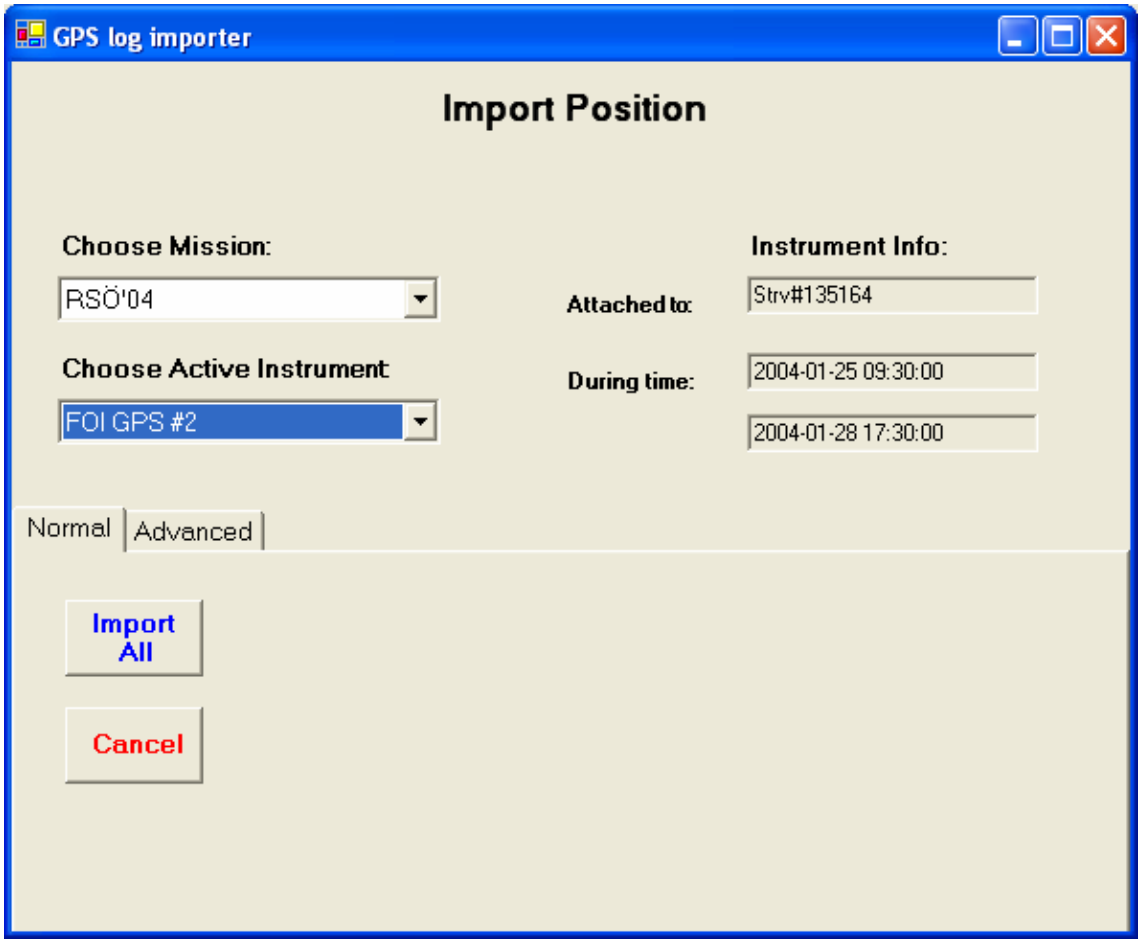


Figure 22 *Importing positions with the GPS log importer.*

7.1.1 Implementation

The GPS log importer is implemented in C# .NET using the application framework. The implementation consists of a module in the user centric tier containing algorithms for parsing coordinate log files in XML and a user module containing the GUI by which the operator interacts.

7.1.2 Usage

When the GPS log is started the user selects a file containing the data to import. After this the user is presented a simple graphical view, in which the user may select a GPS receiver to which the data will be attached. To make it easier for the operator to know which GPS receiver to chose, the current holder of the selected instrument is presented in a separate textbox. After this setup the operator may choose to import all log entries from the file or select those that fall within a certain time span.

7.1.3 Results

The GPS log importer tool is very useful to test the benefits of using the application framework; we built the application very quickly and were able to quickly find, and correct, some weaknesses of the framework. We also found that many of the data types in the DataType module where rather impractical to use. A direct consequence of this is that we introduced data type helpers which are functions that may be used to simplify the use of some complex data types. Example functions are the RT-90 and WGS-84 coordinate constructors which simplify the creation of such coordinates.

7.1.4 Extensions

We have identified many useful extensions to this tool that would increase the usefulness of it. Allowing the operator to add offsets and manipulate the data before importing it is one of the most simple and important extensions. Adding support for filtering data based on certain criterions and to simplify the user interface are others. Another possibility is to integrate data import functions like this in the mission setup tool, see chapter 7.2, or add the mission tree structure in a view which would make it easier to relate positions to the right object.

7.2 Mission setup tool

The mission setup tool is used to create conceptual models and instrumentation plans for missions. These are usually set up prior to a mission and form the basic structure of the organization behind an operation.

While the tool is a useful product in itself it also provided an excellent way of testing complex relations in the data model and how the object oriented mapping in the framework worked in reality. The process of implementing this tool led to several redesigns of the framework.

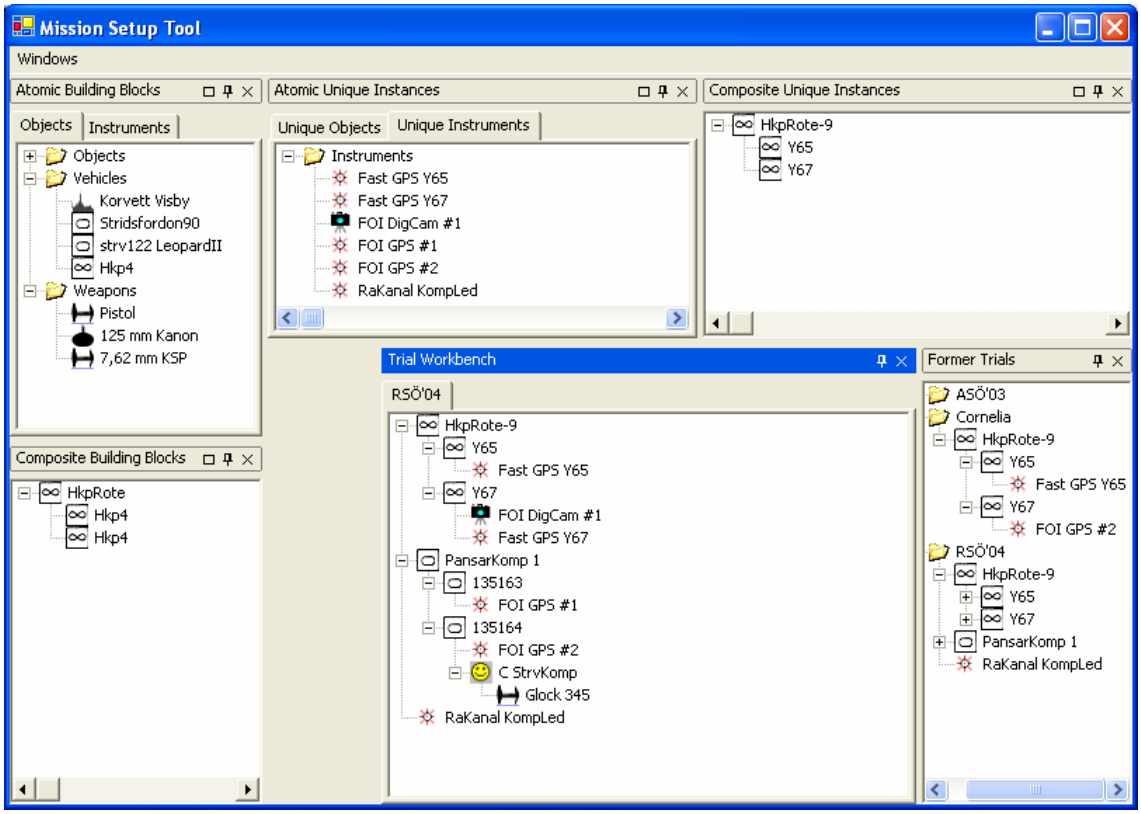


Figure 23 *Modeling a fictive operation with the mission setup tool.*

7.2.1 Implementation

The mission setup tool is implemented in C# .NET using the application framework described in chapter 4. The implementation consists of a user module containing the GUI through which the operator interacts. All interactions are passed on to a second module in the user centric tier which interacts with the proxy. This architecture makes it easy to change GUI since all methods for manipulating data are found in the lower layers.

The GUI consists of six independent forms which use the docking capabilities found in many Windows applications. The mission setup tool was developed mainly as a tool to verify the data model and the application framework. As a stand-alone application it is not considered complete. For example, not all of the inherited objects classes are implemented in the Atomic Building Blocks form and the use of dynamic attributes is not supported.

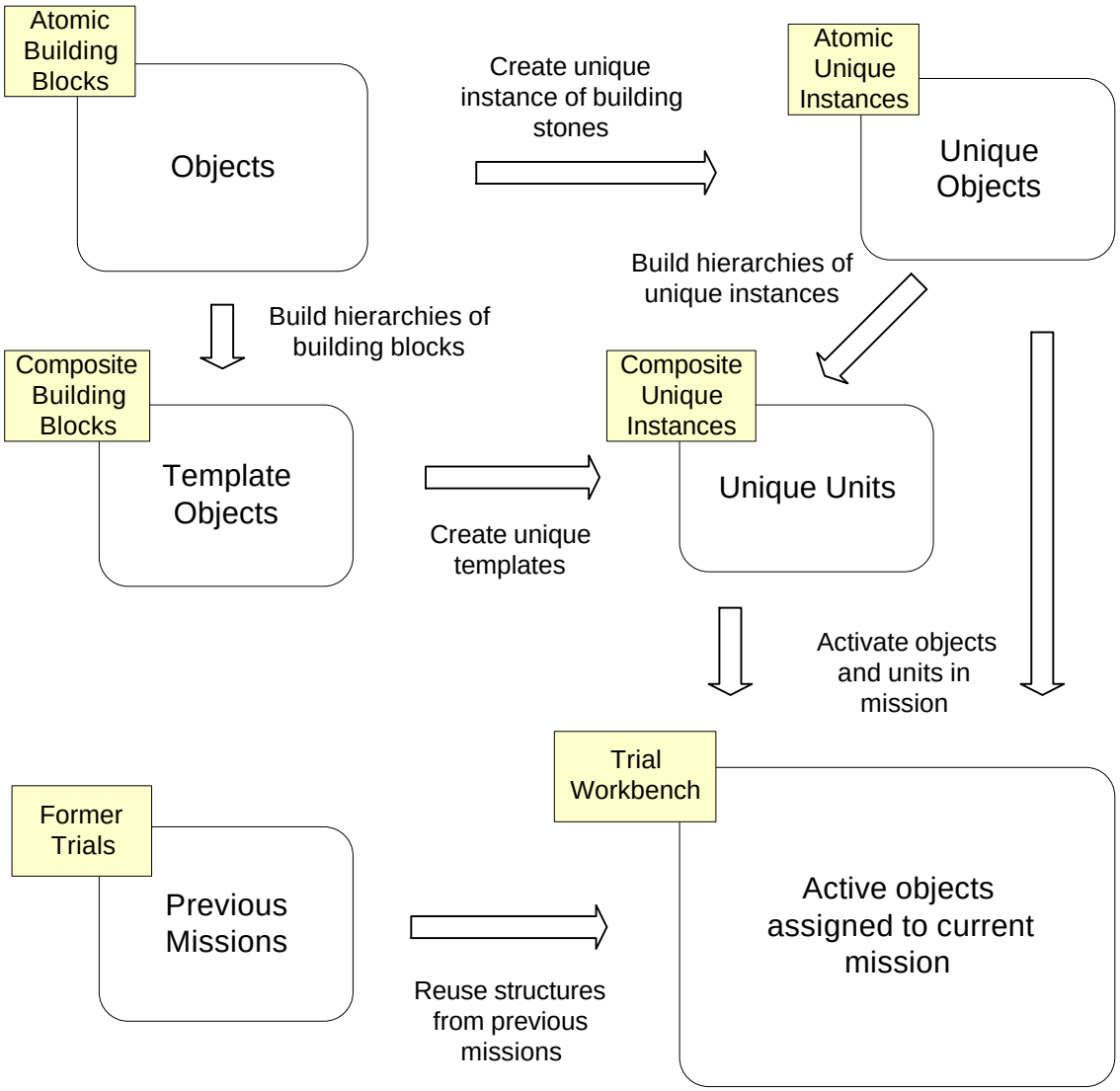


Figure 24 *Workflow of building a conceptual model with the Mission setup tool.*

7.2.2 Usage

When the mission setup tool is started the six forms described in figure 24 are displayed in a large window with docking capabilities. These six forms are logically and functionally separated in different stages of the setup phase. The Atomic Building Blocks form is used to create single objects see chapter 2.3.1. The Composite Building Blocks form is used to create complex template objects, see chapter 2.3.2, by dragging and dropping objects from the Atomic Building Blocks Form. The Atomic Unique Instances form and the Composite Unique Instances form are used in similar ways. The difference is that in these forms you work with objects and templates that are unique instances, see chapter 2.3.5. Together these

four forms are used to design and initialize components in the modeling domain block that are used to create the conceptual model.

The Trial Workbench is the form where the actual mission setup is made, or where the conceptual model and instrumentation plans are put together. A mission model can be composed out of components from old missions or designed from scratch using the available components in the modeling domain block. In either case the mission is setup by dragging and dropping unique objects or units from their respective form to the Trial Workbench. The contents of the missions are then considered activated, see section 2.4.1.

7.2.3 Results

The direct result of building the mission setup is one single useful tool handling all work needed to prepare the database for import of data collected during a mission. As a deliberate side effect, the tool is also used to model the participating objects in a mission and the hierarchal command structures different units have or the complex structures of some objects, for example naval ships.

The tool is also very useful to test the frameworks effectiveness regarding complex relations. Our tests led to some redesign of the code generation of lower tiers of the framework, with modifications that greatly enhanced usability.

Another important result is the discovery of the performance problem that arises when handling multiple objects dependent on information from the database. At startup the tool fetches information about all objects in the database one by one. This is very inefficient and slows down performance. Chapter 9 describes a few suggested solutions to this problem, which must be solved.

7.2.4 Extensions

The Mission setup tool can be extended in many ways. The most important extension we have identified is to add some kind of hierarchical ordering system in the different forms. When the database grows large it is apparent that the information displayed in the different forms also will grow dramatically and it will become more difficult to get a good overview of the contents of the database. We therefore identified a need to organize the objects even further. Discussions on this matter have led to several

suggestions on how to deal with this problem. Whichever is chosen it is clear that it will improve usability of this tool.

The most straightforward solution we found was to introduce a folder system where each object may belong to a certain folder. The Apache helicopter, for instance, is a Vehicle which may be organized into the attack helicopter folder. This gives us two possible ways to sort the helicopter, either as a vehicle or as an attack helicopter. A problem to solve is how to store this folder structure. One solution would be to keep this centralized at the database level by introducing another relation between objects and folders. This solution would make the database even more complex and the only benefit gained is that it will become easier to find objects in tools like the setup tool. It can be argued that this information should be stored at an application level since it is not clear that every application wants to sort the objects on the same criterions.

Another way to solve the problem of too much data displayed is to introduce parameters holding information about how old and how interesting an object is. Very old or uninteresting objects could then be filtered out if the operator knows that he wants to use only common objects or, if the operator doesn't want to filter them out completely, they may be displayed at the bottom of the view. Again there is the question whether this information should be bound to a specific application or if it should be introduced in the database.

A third solution is to introduce the domain concept in the database. By assigning each and every item in the database to a domain, such as fire rescue items, we could filter the objects based on which domain the current mission belongs to. It is not clear how useful this would be though, as one instance of the database is likely to be used within just a few domains.

The second problem we found was that it is very easy, perhaps a little too easy, to create relations between entities in the database using this tool. The user is allowed to create many strange relations which would be logically wrong in reality. In these cases constraints on the relations should be introduced. The most apparent example is to regulate which time objects can be related to another object to prevent them from being allocated multiple times in overlapping time periods. While we want to keep the data model unrestricted perhaps the restrictions should be brought into this tool instead to prevent illegal use of the relations.

Another possible extension is to merge this application with data import applications like the GPS log importer. It would be very convenient to be able to import the log file to a certain GPS simply by right-clicking it and select ‘import’ from the menu. An extension like this can easily be implemented since all the functionality for the import already exists in the GPS log import module in the User Centric Tier.

Chapter 8
Revisions to the MIND framework

The MIND framework was developed to reconstruct and explore DTOs at the Swedish Defense Research Agency. As stated in the introduction, MIND is a component-based visualization framework that integrates domain models, data sources, data converters and presentation views. MIND has been used to demonstrate the applicability of methods and tools in several different domains including combat training with the Swedish Army, naval operations with the Swedish Navy and Rescue operations with the Swedish Rescue Services Agency and Linköping Fire Rescue Department. (Morin, 2002a)

Because the MIND framework has been under development for such a long time and has been tested thoroughly it provides an excellent testing platform for the exploration part of the application framework described in chapter 4. Establishing a connection to MIND was therefore a much prioritized part of developing the application framework, see chapter 6.

Since MIND has been a research project developed under tight time constraints and with the focus on overarching research questions, there has never been time to create an official document describing the architecture or design of the system. A significant part of the work of integrating the two systems has been to get an understanding of the MIND architecture and design, which could be done thanks to the well commented source code and access to an unpublished paper describing the main concepts of MIND (Axelsson, 2001).

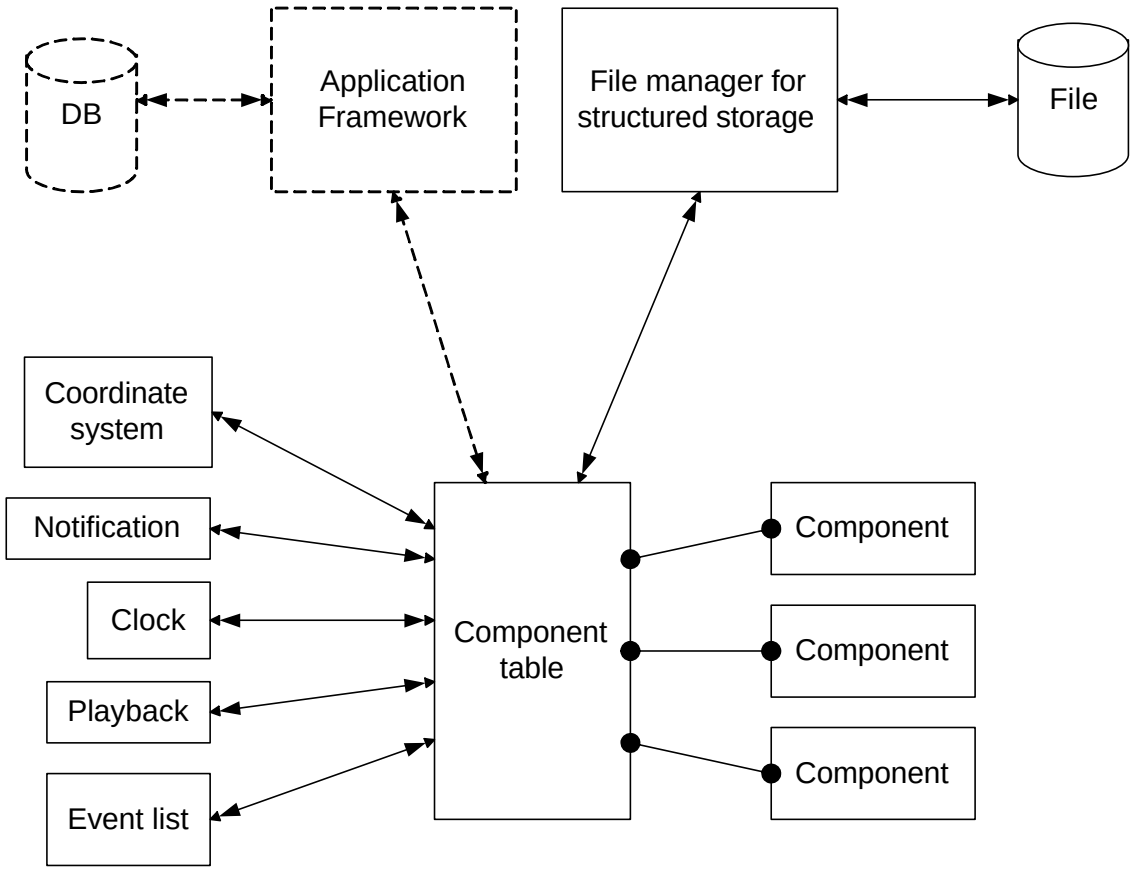


Figure 25 *Schematic drawing of the MIND core with the added connection to the application framework and the database. This drawing is based on a drawing in an unpublished guide to MIND (Axelsson, 2001).*

In order to connect the application framework to MIND we had to make some modifications to the MIND core. As can be seen in figure 25 the component table is very central to MIND. The table contains all the MIND components and is the main connection point for storage and playback. In order to create components based directly on data from the database the natural approach is to extend the model as the dotted lines in the figure indicates. To achieve the connection between the component table and the application framework the component table was extended with functionality to interpret COM objects generated by the application framework. These COM objects are then interpreted by the component table and converted into standard components. Note that the Component table is written in Visual C++ and uses COM to communicate with the different modules shown in the drawing while the application framework is written in C#.NET. Thus the framework must support COM/.NET

interoperability, by generating COM compatible assemblies for the .NET objects as described in appendix B.

When saving scenarios explored using the MIND system, the current approach is to use a structured storage file. This file is mission specific and needs to be set up from scratch every time. Also, the storage format is very specific to MIND, since every object is stored as a binary snapshot of the current state the object is in, which makes the data difficult to use in other applications. The structured storage solution is closely tied to the current implementation of MIND, and future upgrades to the core might make old data difficult to use.

Jenvald proposes a database in which the data is stored and retrieved using a SQL interface (Jenvald, 1996). Storing data in the database would give the benefit that any application can access and interpret the data via a standard SQL interface instead of reading serialized Visual C++ objects as is the case in MIND. Although not implemented, we suggest that the application framework is used for this purpose as well, leaving only data very specific to the MIND application to be stored in a separate file.

The modified version of MIND is incapable of importing the symbols stored in the database since the symbol tables are coded into the program and can not be changed at runtime. Until this has been solved, the user must select an icon manually for each object that needs to be visible in the map views. Future expansions to MIND might also allow the user to select the level of detail on the data to be imported based on a predefined set of criterions.

Chapter 9
Future work

The data model and the application framework presented in this thesis are not intended to be complete in any way. There are many ways to extend them and make the system better. In this chapter we describe a few of the possible enhancements that we found desirable.

9.1 Database

Although we have tried to cover as many special types of data as possible we know for a fact that the data model is not complete. We have discovered several ways to enhance the database, both in terms of what data can be stored and in terms of efficiency.

9.1.1 Multilingual strings

There are many strings stored in the database which are used to display information of some sort. Currently they are stored as regular ANSI strings. In order to change the language of the strings the user needs to copy the entire database and edit the strings manually. A better solution might be to have a global variable in the application framework which selects a language, and replace all strings with a unique identifier. Then the global variable is used to select from which table the string with the unique identifier is to be selected.

figure 26 shows a suggested implementation of these tables along with the relations to the object table. Every table containing strings will have this kind of relation to the string table.

It should be noted that this solution will cost some performance, although it will be very useful when demonstrating the system in different languages, since all presentations can then operate on the same set of data.

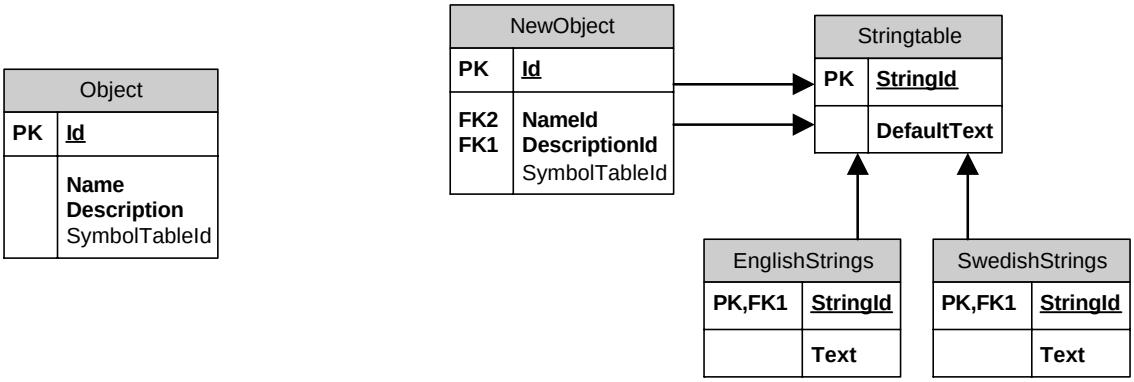


Figure 26 *Left: The current Object model. Right: The Object model with the proposed string table extension. A global variable should be used to identify which string table to use when looking up the name or description of a NewObject. Should the string not be implemented in that table, then the value could be selected from DefaultText instead.*

9.1.2 Time

The time table is very central and will grow rapidly. The table is therefore likely to become slow and a bottleneck of the system. A way to work around this is to remove the table completely and replace all relations to the table with the actual timestamp itself. One timestamp can thus exist in many tables at the same time. Whether this solution will give any benefits is not clear, but it might be something to try out if performance becomes a problem. A negative side effect is that it will become more difficult to select actions based on time, for instance filter out every event that somehow is related to a certain time interval.

9.1.3 Targets

Targets are currently represented as any other object or data in our model. There are however several problems with this representation. One problem is that we seldom know the exact real-world position of the target, there is always an error in measured data. This means that when we have two radar stations measuring the same target, they will differ in their output. The consequence of this is that it is impossible to know when two targets are in fact the same. How do we know which object to relate the detected target to?

There may also be a problem in knowing whether two subsequent blips on a radar screen in fact belong to the same target. A workaround for this is to represent all targets simply as data and let the operator decide how to deal

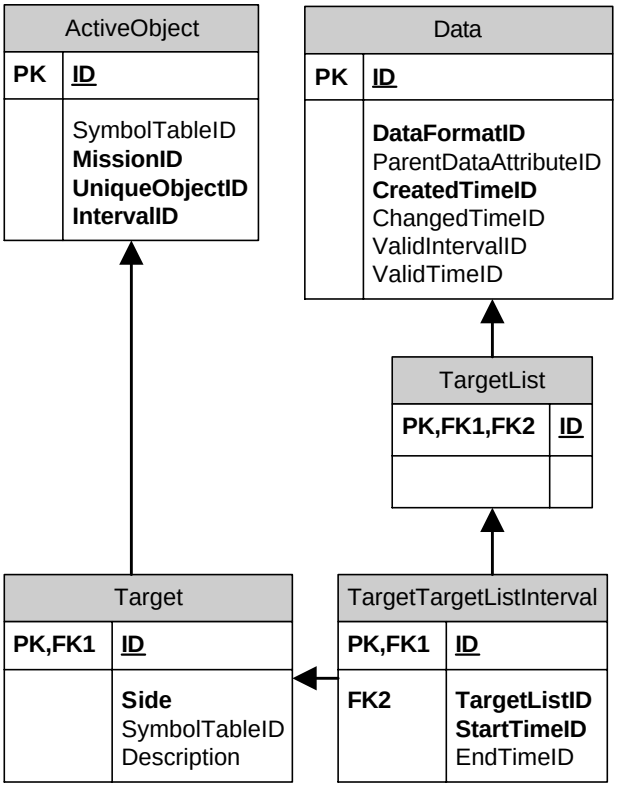


Figure 27 *An extension to the current model of Data and ActiveObject which will allow us to represent the targets in lists of measured target points. Each target may then be related to the actual object that it represents in case the operator knows the ground truth of the target.*

with them. This solution will work fairly well in simple scenarios, but there are also scenarios where we need to represent the actual airplane that the radar is following as well, and when we want to know how different radar stations perceive the same target. The situation may become very complex and an automated way to represent targets could simplify our work. The question remains; what is the best way to represent targets?

It is fair to assume that a special representation of targets is necessary. We have outlined a simple solution using target lists and targets which may or may not be related to objects in our database. This sample solution can be found in figure 27.

9.1.4 Splitting the database

The database can be roughly divided into two parts; the static part which consists of building blocks for setting up a mission and the more dynamic part containing the data collected during and after the mission. The database could therefore be split into two databases to reduce the size of the operating database. It is likely though that the data partition will become

much larger than the building block partition, so it is not certain that the benefits gained from this partitioning outweigh the costs.

9.1.5 Perform tests using other database management systems

For this project Microsoft SQL Server 2000 has been used exclusively. The application framework has been written to allow any back-end DBMS to be used with only minor modifications to the framework. Although not verified we believe the workload of testing another RDBMS or ORDBMS, such as Oracle, would be quite limited. Of extra interest might be to evaluate specialized temporal and spatio-temporal database management systems. Even though we have decided not to evaluate them further since we could model the data with ordinary relations, it might become interesting to verify whether these systems could help improve performance or usability.

9.1.6 Dynamic attributes

The object model is currently based on inheritance. A vehicle, for instance, is an object extended with some extra features. Since traditional RDBMS such as SQL Server does not allow inheritance, this is implemented simply as a 1-1 relation between an object and a vehicle where the vehicle has a foreign key to an object; this key also serves as a primary key for the vehicle. This solution simulates inheritance pretty well. In a similar way we have created solid objects, person objects, map objects and equipment objects.

Still, it is impossible to cover every possible extension to the object model, and is also not very hard to imagine that for some special vehicles the vehicle model is not adequate. We might for instance want to store the engine capacity of a large truck, or the number of wheels it has. To allow this we also need some sort of dynamic attribute model, where any attribute-value pair can be added to any object.

It can be argued whether there really is any need to organize the objects into a hierarchical inheritance structure when we still need these dynamic attributes. Is it not better to make all objects just objects, with dynamic attributes where necessary? This model would certainly be more general and consistent, but the drawbacks would be that we need to store more data (we would have to store what kind of attribute is stored for every single attribute-value pair). Also, searching the database using dynamic attributes is much slower than searching a column in an ordinary table. Therefore we

believe that the current model is better, but it would be nice to test how much difference it will make in terms of speed and memory.

9.1.7 Evaluating object oriented database manager systems

As can be read in section 3.2, the ODBMS are catching up on the RDBMS in terms of maturity. The vendors state that their databases operate at much higher speeds compared to traditional RDBMS. These statements have not been verified and the flexibility of the ODBMS is not known either. Heavy testing and evaluation of ODBMSs are desired to find out whether they are applicable for this kind of framework.

9.2 Application framework

The application framework is not very optimized at all, nor is it complete in terms of functionality. This section tries to sort out what kind of optimizations and modifications we suggest as future work on the application framework.

9.2.1 Database mediator

Today the database proxy is accessing the database directly via OLE DB. With a solution like this we automatically inherit all the weaknesses of OLE DB, such as its inability to work in connected mode. It is also very difficult to handle concurrency conflicts in a well organized manner. Right now the framework simply changes the underlying data or generates exceptions when two different applications operate on the same data.

Writing a database mediator (DM) that communicates with the database is perhaps a better alternative. With a solution like this we can have every front-end application request data from the DM and we get much better control over the underlying engine. We can for instance refuse access for one certain application before another application releases it (much like transactions, but on a larger scale).

We can also have the mediator send events to the listening clients when a certain piece of data is altered. This solution will certainly improve performance a lot since we will not need to update the entire table every time we load a piece of data to verify its correctness.

9.2.2 Data type helpers

Some of the data types are very complex and to generate standard data, such as WGS-84 coordinates, it is convenient to have helper functions that simplify the process. This kind of helper functions are implemented in some cases, but far from all. To make the framework really easy to use, all data types should be thoroughly evaluated and helper functions added if necessary.

9.2.3 Application adapted database proxies

Today the database proxy fetches records directly from the database, every time they are accessed, to allow a multi user environment. This causes a lot of traffic on the network and reduces performance of some applications. In some cases performance is more vital than concurrency control. In these cases the applications could use a different proxy which fetches data from the database less frequently and holds the information internally. This would reduce network traffic and enhance performance. However this proxy is vulnerable to multi-user environments, if several users change information on the same object simultaneously.

9.3 Applications

Today only three applications exist that interoperate with the application framework, the Mission setup tool, the GPS log importer and the extended MIND. MIND is described more in the next section, in this section we try to identify a few of the applications we believe are natural to develop as a next step in this project.

9.3.1 Miscellaneous log import tools

We need tools for importing the data log files that are currently used in MIND. Example data are pictures, video and audio. Currently there exists a module for importing pictures via the PIX log files and one for GPS log files, but there are numerous more that MIND can handle, and so should the application framework. As for the coordinate log importer, it should be extended with additional functionality and features to improve usability, such as importing several log files in one click.

One possible alternative is to develop several user centric modules for importing various types of data and then tie them all together in one single graphical application.

9.3.2 Metadata workbench

A metadata workbench was developed for MIND (Albinsson et al, 2004) to create couplings between metadata and the original raw data. Using this tool the operator can relate a photograph to the people on the picture or a video recording to the actors in it. One of the major strengths and original motivations of creating this database framework was the increased ability to create relationships between objects, data and metadata. Bearing this in mind the metadata workbench should be upgraded to make use of the database.

9.3.3 Tools for communication analysis

There is some support for analyzing communication in MIND (Albinsson, Fransson & Morin, 2003). With these tools the operator can enter who the actors are in a communication segment. This data may then be presented in the MIND visualization framework by creating a view. These tools could be given an extra dimension if they were used in conjunction with the database since it would then be possible to create relations between the actors in the recorded communication clip and the actual clip.

Another tool could be implemented to do noise reduction and filter out silent sessions that have been recorded. The database model supports this, yet there are no tools that currently use it.

9.3.4 Data displayer

There is lots of data in the database, and not even MIND is capable of displaying it all, so writing a few small applications for displaying data in various ways might be desirable. Such a tool could also be written as a component for integration with MIND.

9.3.5 Symbol table utility

A utility for organizing symbol tables and symbol libraries as well as adding, creating and removing symbols would be very handy for operating on symbol tables.

Some possible features of the tool would be to create libraries of symbols, organize these libraries and symbols, edit the symbols, add descriptions to the symbols and create tables consisting of a set of symbols that can be assigned to any object.

9.3.6 Object designer

The database allows just about any type of object to be created, but it takes some work to create it by hand. Therefore it would be excellent to have a graphical tool for designing new objects with custom attributes.

9.3.7 Export tools

Exporting the data from a mission to several other formats has been discussed. Some examples are: PowerPoint, HTML, XML and executables. Creating such a tool is probably not an easy task since it is not clear what data should be included in the export. However, a tool that is configurable so that the operator may select what is to be presented in the exported data and to which format might become very useful.

9.4 MIND

MIND is currently the only supported export format. The data is imported from the database via a graphical interface in MIND. The project of integrating the database in MIND has only just begun and there are several paths to continue this integration. No matter which of the solutions is selected, some remakes need to be done in the kernel to support our dynamic active object model where objects may be aggregates of other objects. This is not allowed in the MIND core unless special components are created for that purpose.

9.4.1 Add support for more object and data types

The first alternative is to continue on the chosen path and implement support for more kinds of objects and data types. This should not be too hard for the basic structures that map well to the components in MIND today. Structures that do not map well to the existing MIND components should perhaps be given their own components, a solution which requires more work.

9.4.2 Convert MIND source code to .NET

Another alternative is to make a port of MIND in .NET and meanwhile redesign the MIND framework for a better match with the database. This job is of course very expensive and takes quite some time to perform because of the size of MIND. Bearing in mind that this system is to be continued, this idea should be thoroughly considered. Maintaining the

system will become much easier if the frameworks get a common architecture that fit well together.

9.4.3 Global relations

Every view in MIND defines its own relations between the incorporated objects. It would be preferable if the relations were made global so that a connection made in one tool could be interpreted by any other tool. Currently the MIND core does not support this, but with the use of the database this feature should not be too difficult to implement in the MIND components.

One example of this is metadata which may be inserted via the metadata tools available in MIND. The user may for instance enter text describing a vehicle in a photograph, but there is currently no way to connect that photograph to the representation of that vehicle in MIND. Using global relations between objects and data would give this possibility.

9.4.4 Extended database support

Currently the implemented changes in MIND allows only importing data from the database. Adding support which will allow the operator not only to import data, but also to export changes made in MIND would greatly improve the interoperability between the two frameworks.

Chapter 10
Summary and Conclusions

10.1 Summary

The main purpose of this report is to present a centralized storage facility based on a DBMS that aims to replace the current serialized single file storage structure of the MIND mission history data. During the development process we investigated data storage modeling needs for reconstruction and exploration of DTOs and the requirements this model puts on a DBMS. To interact with the model and the storage system, user oriented applications are needed. To facilitate development of the applications and standardize the interface between applications and the database we have developed an application framework.

10.2 Data model

The data model we present supports most of the features that the MIND system has today. It is constructed in a general manner allowing future extensions with none or few modifications. The model supports handling data from previous field trials as shown in the informal test runs. However, as the complexity of the mission history is increasing, future use will surely put emphasis on new areas. Therefore future extensions and modifications are very likely and the model should not be thought of as complete. It should rather be used as a base for further studies which might eventually lead to a satisfying model. It is also a fact that some areas of the model need more thorough investigation. This is especially true concerning the analysis parts, that is the adding of metadata and classifications, to which we have not been able to build test applications due to the time constraints of this project.

10.3 Database management system

The complexity of storing a reconstruction of a DTO put high demands on the DBMS. The DBMS should be able to quickly sort and query several tables with millions of records as well as tables containing very large files,

e.g. audio and video data. We have investigated some of the most well known RDBMSs as well as some more modern object oriented systems. We believe that most of these DBMS are capable of handling the complexity of the data model as well as the probable storage need of the mission history data. Since the technical differences regarding the specifications we have investigated are small, we believe that the DBMS choice used in this data storage facility should be based on which product the database developers and administrators have access to and are used to work with rather than technical details. However the database system chosen should be thoroughly tested with an extreme amount of records. There may be performance differences among the systems when managing large amount of records.

10.4 Application framework

Using the programming framework together with the model is an attractive way to get the objects and relations you need to build effective applications. However, in some cases several consecutive relations in the model make the programming interface complex. In these cases, should modification of the data model be needed, the framework is very flexible and allows changes with only minor updates. Furthermore the framework also supports a swift switching of the underlying database engine.

The application framework of today has some deficits. The most urgent issue to deal with is performance. The proxy module should be revised to decrease network traffic.

10.5 Application development

To fully be able to reconstruct and explore a DTO we need tools for building conceptual models and instrumentation plans, import collected data and finally explore and analyze the composed mission history. The application framework facilitates the task of creating these applications. A programmer hardly needs any knowledge of the DBMS to be able to create an application that operates on the database.

Using two high-level applications that we developed, one for the modeling part and one for the data collection part is a good way to test different data structures. These applications are important in the iterative design process since they help us to enhance the data storage model and discover problems

in the application framework. In addition to this the modification of the existing MIND framework makes us able to explore missions stored in the database.

Currently only a subset of the data model is supported by MIND and no other exploration application has been written, so more effort needs to be put into the MIND export module before the framework can be used to explore more complex missions.

10.6 General conclusions

Considering the complex data model and the desired expansions to MIND we believe that the database approach and the data model suggested are suitable for storing mission histories for DTOs. However the system must be carefully optimized for applications where speed is important. The system presented in this report is today ready to be used for minor operations when there are no time constraints creating the conceptual models and instrumentation plans. Since we do not consider the application framework to be capable of handling real time streaming data at this moment, we suggest further investigation in this area. We also suggest that the framework and data model are extended as needed when new applications are designed. Extending the MIND framework to include a higher level of support for the database connection would also be a natural way to continue this work.

Chapter 11
References

Albinsson, P. & Fransson, J. (2001). *Communication visualization - an aid to military command and control evaluation*. In Proceedings of the 45th Annual Meeting of the Human Factors and Ergonomics Society, October 8-12, Minneapolis/St. Paul, USA.

Albinsson, P., Fransson, J. & Morin, M. (2003). *Finding information needs in military command and control systems using exploratory tools for communication analysis.* In Proceedings of the 47th Annual Meeting of the Human Factors and Ergonomics Society, October 13-17, Denver, USA.

Albinsson, P., Morin, M. & Thorstensson, M. (2004). *Managing metadata in collaborative command and control analysis.* To be presented at the 48th Annual Meeting of the Human Factors and Ergonomics Society, September 20-24, New Orleans, USA.

Axelsson, M. (2001). *MIND.* Unpublished.

Boehm, B. (1986). *A Spiral Model of Software Development and Enhancement*. ACM SIGSOFT Software Engineering Notes, August 1986,

Bohuszewicz, K., Czyowicz, M., Janik, M., Jarosz, D., Mazan, P., Mierzejewski, M., Olszewski, M., Peryt, W., Radomski, S., Szarwas, P., Traczyk, T., Tukendorf, D., & Wojcieszuk, J. (2003). *Comparison of Oracle, MySQL and Postgres DBMS.* Published on the Internet: http://det-dbalice.if.pw.edu.pl/det-dbalice/ttraczyk/db_compare/db_compare.html

Chigrik, A. & Vartanyan, (2004) S. *SQL Server Articles – Comparison.* Published on the Internet: http://www.mssqlcity.com/Articles/Compare/Compare.htm

Elmasri, R. & Navathe, S. (2000) *Fundamentals of database systems.* Addison-Wesley. ISBN 08-0531-755-4.

Fermi National Accelerator Laboratory/Computing Division (2003). *MySQL General Information – Comparison of Oracle, MySQL and*

PostgreSQL DBMS. Published on the Internet: http://www-css.fnal.gov/dsg/external/freeware/mysql-vs-pgsql.html

Jensen, C.S. (2000). *Temporal Database Management.* Published on the Internet: http://www.cs.auc.dk/~csj/Thesis/.

Jenvald, J. (1996). *Simulation and Data Collection in Battle Training.* Linköping Studies in Science and Technology, Thesis No 567, Linköping University, Linköping, Sweden.

Jenvald, J. (1999). *Methods and Tools in Computer-Supported Taskforce Training*. Linköping Studies in Science and Technology, Dissertation No. 598, ISBN 91-7219-547-9, Linköping University, Linköping, Sweden.

Microsoft Inc. (2004). *SQL Server Architecture Maximum Capacity Specifications.* Published on the Internet: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/architec/8_ar_ts_8dbn.asp

Moniz, J. (1999). *Enterprise Application Architecture.* Wrox Press Inc. ISBN 18-6100-258-0.

Morin, M. & Thorstensson, M. (2000). *Cornelia Datainsamlingsplan.* Unpublished internal report (In Swedish).

Morin, M. (2002a). *Multimedia representation of Distributed Tactical Operations.* Department of Computer and Information Science Dissertation No. 771, Linköping University, Linköping, Sweden.

Morin, M. (2002b). *Modeling Distributed Tactical Operations for Command and Control Analysis.* In Proceedings of the Swedish-American Workshop on Modeling and Simulation, SAWMAS-2003, 2002, Orlando, USA.

Morin, M., Jenvald, J. & Thorstensson, M. (2000). *Computer-supported visualisation of rescue operations.* Safety Science, 35, 3-27.

Morin, M., Jenvald, J., Nygren A., Axelsson, M. & Thorstensson, M. (2003). *A study of first responders' use of digital cameras for documenting rescue operations for debriefing and analysis,* In Proceeding of the International Emergency Management Society's Tenth Annual Conference, TIEMS 2003, June 3-6, Sophia-Antipolis/Nice, France.

NATO (2002), *The Land C2 Information Exchange Data Model, Working Paper 5-5, Edition 5.0, 18 March 2002.* ATTCIS WG, SHAPE, Belgium.

Thorstensson, M. (2002a). *Rapportering av genomfört fältförsök med FM Helikopterflottilj*. FOI Memo 02-2918. (In Swedish)

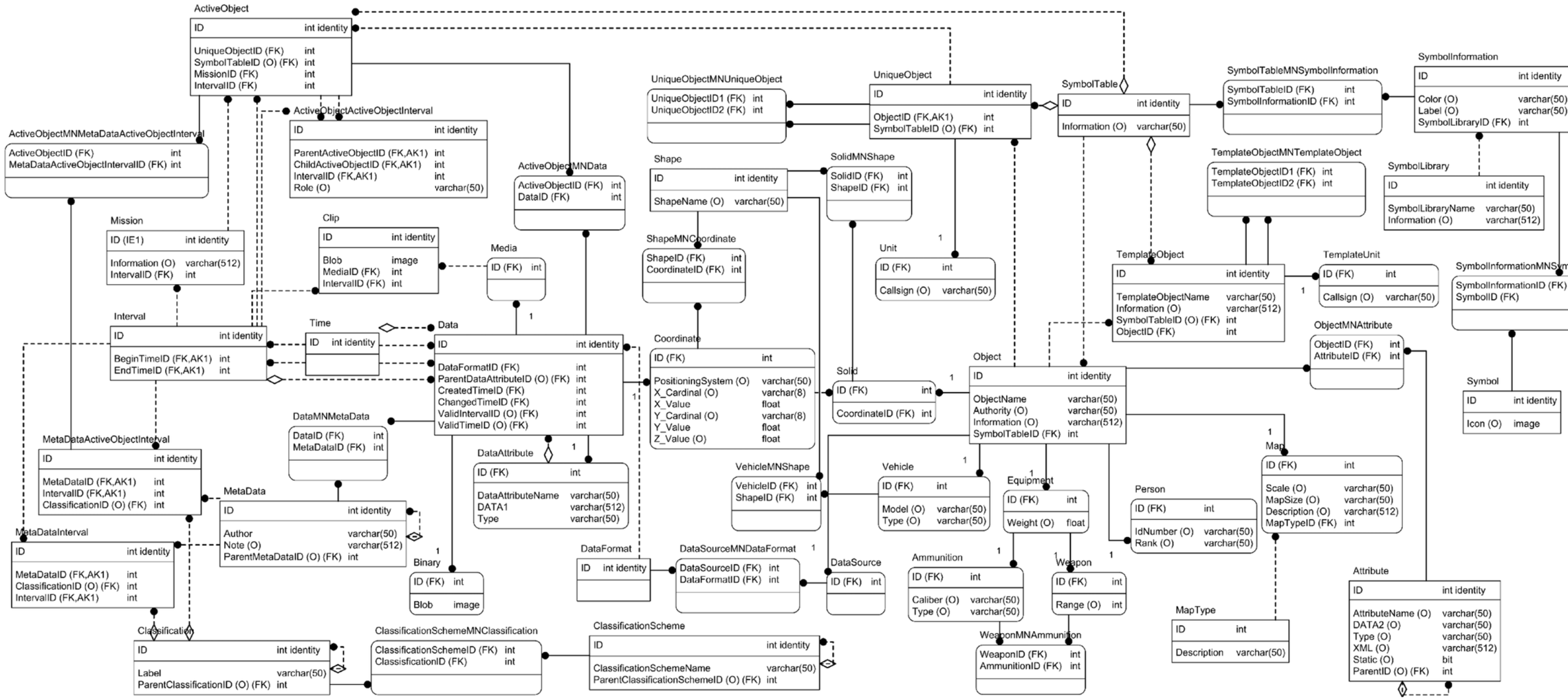
Thorstensson, M. (2002b) *Data Collection in Rescue Operations*. In Proceedings of the International Emergency Management Society's Ninth Annual Conference (TIEMS 2002). May 14-17, 2002, Waterloo, Canada.

Thorstensson, M., Björneberg, A., Tingland, B. & Tirmén Carelius, M. (2001). *Computer-Supported Visualisation of an Inter-Agency Exercise in the Stockholm Underground*. In Proceedings of The International Emergency Management Society's Eighth Annual Conference (TIEMS 2001). June 19-22, Oslo, Norway.

Thorstensson, M., Jenvald, J., Morin, M. (2002) *Modeling and visualization of naval units.* Swedish Defense Research Agency. Report No. FOI-R--0524--SE.

Appendix A
Data model for reconstruction and exploration of DTO

The data model for reconstruction and exploration of Distributed Tactical Operations is enclosed on the next page.


ActiveObject
ID int identity
UniqueObjectID (FK) int
SymbolTableID (O) (FK) int
MissionID (FK) int
IntervalID (FK) int
ActiveObjectActiveObjectInterval
ID int identity
ParentActiveObjectID (FK,AK1) int
ChildActiveObjectID (FK,AK1) int
IntervalID (FK,AK1) int
Role (O) varchar(50)
ActiveObjectMNMetaDataActiveObjectInterval
ActiveObjectID (FK) int
MetaDataActiveObjectIntervalID (FK) int
ActiveObjectMNData
ActiveObjectID (FK) int
DataID (FK) int
UniqueObjectMNUniqueObject
UniqueObjectID1 (FK) int
UniqueObjectID2 (FK) int
UniqueObject
ID int identity
ObjectID (FK,AK1) int
SymbolTableID (O) (FK) int
SymbolTable
ID int identity
Information (O) varchar(50)
SymbolTableMNSymbolInformation
SymbolTableID (FK) int
SymbolInformationID (FK) int
SymbolInformation
ID int identity
Color (O) varchar(50)
Label (O) varchar(50)
SymbolLibraryID (FK) int
TemplateObjectMNTemplateObject
TemplateObjectID1 (FK) int
TemplateObjectID2 (FK) int
SymbolLibrary
ID int identity
SymbolLibraryName varchar(50)
Information (O) varchar(512)
Shape
ID int identity
ShapeName (O) varchar(50)
SolidMNShape
SolidID (FK) int
ShapeID (FK) int
ShapeMNCoordinate
ShapeID (FK) int
CoordinateID (FK) int
Unit
ID (FK) int
Callsign (O) varchar(50)
TemplateObject
ID int identity
TemplateObjectName varchar(50)
Information (O) varchar(512)
SymbolTableID (O) (FK) int
ObjectID (FK) int
TemplateUnit
ID (FK) int
Callsign (O) varchar(50)
SymbolInformationMNSymbol
SymbolInformationID (FK) int
SymbolID (FK) int
Mission
ID (IE1) int identity
Information (O) varchar(512)
IntervalID (FK) int
Clip
ID int identity
Blob image
MediaID (FK) int
IntervalID (FK) int
Media
ID (FK) int
Interval
ID int identity
BeginTimeID (FK,AK1) int
EndTimeID (FK,AK1) int
Time
ID int identity
Data
ID int identity
DataFormatID (FK) int
ParentDataAttributeID (O) (FK) int
CreatedTimeID (FK) int
ChangedTimeID (FK) int
ValidIntervalID (O) (FK) int
ValidTimeID (O) (FK) int
Coordinate
ID (FK) int
PositioningSystem (O) varchar(50)
X_Cardinal (O) varchar(8)
X_Value float
Y_Cardinal (O) varchar(8)
Y_Value float
Z_Value (O) float
Solid
ID (FK) int
CoordinateID (FK) int
Object
ID int identity
ObjectName varchar(50)
Authority (O) varchar(50)
Information (O) varchar(512)
SymbolTableID (FK) int
ObjectMNAttribute
ObjectID (FK) int
AttributeID (FK) int
Symbol
ID int identity
Icon (O) image
DataMNMetaData
DataID (FK) int
MetaDataID (FK) int
DataAttribute
ID (FK) int
DataAttributeName varchar(50)
DATA1 varchar(512)
Type varchar(50)
VehicleMNShape
VehicleID (FK) int
ShapeID (FK) int
Vehicle
ID (FK) int
Model (O) varchar(50)
Type (O) varchar(50)
Equipment
ID (FK) int
Weight (O) float
Person
ID (FK) int
IdNumber (O) varchar(50)
Rank (O) varchar(50)
Map
ID (FK) int
Scale (O) varchar(50)
MapSize (O) varchar(50)
Description (O) varchar(512)
MapTypeID (FK) int
MetaDataActiveObjectInterval
ID int identity
MetaDataID (FK,AK1) int
IntervalID (FK,AK1) int
ClassificationID (O) (FK) int
MetaData
ID int identity
Author varchar(50)
Note (O) varchar(512)
ParentMetaDataID (O) (FK) int
MetaDataInterval
ID int identity
MetaDataID (FK,AK1) int
ClassificationID (O) (FK) int
IntervalID (FK,AK1) int
Binary
ID (FK) int
Blob image
DataFormat
ID int identity
DataSourceMNDataFormat
DataSourceID (FK) int
DataFormatID (FK) int
DataSource
ID (FK) int
Ammunition
ID (FK) int
Caliber (O) varchar(50)
Type (O) varchar(50)
Weapon
ID (FK) int
Range (O) int
WeaponMNAmmunition
WeaponID (FK) int
AmmunitionID (FK) int
MapType
ID int
Description varchar(50)
Attribute
ID int identity
AttributeName (O) varchar(50)
DATA2 (O) varchar(50)
Type (O) varchar(50)
XML (O) varchar(512)
Static (O) bit
ParentID (O) (FK) int
Classification
ID int identity
Label varchar(50)
ParentClassificationID (O) (FK) int
ClassificationSchemeMNClassification
ClassificationSchemeID (FK) int
ClassificationID (FK) int
ClassificationScheme
ID int identity
ClassificationSchemeName varchar(50)
ParentClassificationSchemeID (O) (FK) int

Appendix B
COM / .NET Interoperability

B.1.1 Using COM objects from .NET with early binding

COM objects are defined by their interfaces and an implementation of the interface. They are generally created by using the *CoCreateInstance* method by specifying what interface to implement and what implementation to use. The term ‘early binding‘ implies that the implementation and its type library are known at development time.

In .NET there is no equivalent to the *CoCreateInstance* method of COM. Instead we need to generate Runtime Callable Wrappers (RCW)[1]. The RCW wraps the COM object and mediates between it and the .NET common language runtime (CLR)[2] environment, making the COM object appear to .NET clients just as if it were a native .NET object and making the .NET client appear to the COM object just as if it were a standard COM client.

The RCW can be created in numerous ways. One way is to use the *TlbImp.exe* tool which *converts* a COM type library to a .NET understandable DLL. With Visual Studio .NET this can also be done by adding a reference to the COM type library. A third way to do it is to write the wrappers by hand using the *DllImport* class.

COM objects are then created just like any other .NET objects by calling the RCW default constructor.

A potential problem with this approach is that COM and .NET objects have different lifecycles. A COM object is immediately destroyed when it is unreferenced but a .NET object is not actually destroyed until the garbage collector collects it. How the garbage collector works is beyond the scope if

---

[1] See URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconruntimecallablewrapper.asp

[2] See URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconthecommonlanguageruntime.asp

this report and for now it is sufficient to know that this can be solved by explicitly calling the garbage collector every time a COM object needs to be released. Although this solves the problem it introduces a large overhead since garbage collecting is a very heavy operation. You may not want to pay the price of garbage collecting the entire system just to release one object. Fortunately there is a function that releases one COM object, *System.Runtime.InteropServices.Marshal.ReleaseComObject*, this may be called instead.

B.1.2 Using COM objects from .NET with late binding

If a COM object supports th*e IDispatch* interface it can be bound late by specifying the *ProgID* or the *CLSID*, or the unique identifiers of the implemented classes, of the implementation at runtime. As this is retrieved at runtime it is not possible to pregenerate a RCW as could be done with the early bound object. The solution to this is to create a *System.Type* using the *Type.GetTypeFromProgID* or the *Type.GetTypeFromCLSID* methods respectively. The COM object can then be created using the *Activator.CreateInstance* method and call a method via the *Type.InvokeMember* function.

B.1.3 Using .NET objects from COM

To let COM interoperate with .NET objects we need a structure called the COM Callable Wrapper (CCW)[3], which wraps a .NET object and mediates between the object and the CLR environment. COM needs every component to have a strong name, so also for the .NET components. It must also exist in the Global Assembly Cache (GAC)[4] or in the applications directory tree. Finally the registry must have entries for the component for the COM client to be able to locate a server when creating an object.

The strong name is typically generated using the sn.exe tool shipped with the .NET Source Development Kit (SDK). This tool generates a file containing the strong name pair which can then be inserted into the assembly to associate the assembly with the strong name.

---

[3] See URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconcomcallablewrapper.asp

[4] See URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconglobalassemblycache.asp

To install the assembly into the GAC you should use the gacutil.exe tool shipped with .NET SDK. This step is only necessary if you need the assembly to be globally available. If not, it is enough to copy the assembly Type Library (TLB) to the applications directory tree. The .NET assembly can then be accessed using the *#import* directive in VC++.

Creating the registry entries is done using the regasm.exe tool from the .NET SDK. This tool reads the Metadata from the class and generates the proper entries in the registry.

To create the .NET component, *CoCreateInstance* can be used just as with any other COM component. A CCW is then created based on the .NET class found in the registry which is associated with the requested *CLSID*. Since COM components are always created using the default constructor, the .NET components must also have one. It is this constructor that is called when *CoCreateInstance* is called. The CCW will take care of any conversions needed between COM natives and their .NET equivalents, for instance between *BSTR* and *System.String*.

Appendix C
Sample GPS receiver log file

```
<Root>
  <Positions>
    <SourceType="Garmin GPS 12 CX">GPS-92-ANJ</Source>
    <Objectid Type="Name">ANJ</Objectid>
    <Sample Date="2003-03-12" Time="13:14:14">
      <Position Type="WGS84">
        <Latitude CardinalPoint="North">58.40749</Latitude>
        <Longitude CardinalPoint="East">15.5189</Longitude>
      </Position>
    </Sample>
    <Sample Date="2003-03-12" Time="13:14:19">
      <Position Type="WGS84">
        <Latitude CardinalPoint="North">58.40752</Latitude>
        <Longitude CardinalPoint="East">15.51877</Longitude>
      </Position>
    </Sample>
    <Sample Date="2003-03-12" Time="13:27:33">
      <Position Type="WGS84">
        <Latitude CardinalPoint="North">58.40763</Latitude>
        <Longitude CardinalPoint="East">15.51891</Longitude>
      </Position>
    </Sample>
    <Sample Date="2003-03-12" Time="13:27:37">
      <Position Type="WGS84">
        <Latitude CardinalPoint="North">58.40766</Latitude>
        <Longitude CardinalPoint="East">15.51894</Longitude>
```

```
      </Position>
    </Sample>
    <Sample Date="2003-03-12" Time="13:27:42">
      <Position Type="WGS84">
        <Latitude CardinalPoint="North">58.4077</Latitude>
        <Longitude CardinalPoint="East">15.51899</Longitude>
      </Position>
    </Sample>
    <Sample Date="2003-03-12" Time="13:27:46">
      <Position Type="WGS84">
        <Latitude CardinalPoint="North">58.40772</Latitude>
        <Longitude CardinalPoint="East">15.51903</Longitude>
      </Position>
    </Sample>
    <Sample Date="2003-03-12" Time="13:27:50">
      <Position Type="WGS84">
        <Latitude CardinalPoint="North">58.40775</Latitude>
        <Longitude CardinalPoint="East">15.51906</Longitude>
      </Position>
    </Sample>
  </Positions>
</Root>
```

Appendix D
Sample PIX log file

```
<Root>
  <RTJInfo>
    <Date>2002-12-11</Date>
    <EditorName>Markus</EditorName>
    <EditorLastName>Axelsson</EditorLastName>
    <EditorSignature>MA</EditorSignature>
    <ReportID>MA</ReportID>
    <CameraID></CameraID>
  </RTJInfo>
  <Reports>
    <Observer>Markus Axelsson</Observer>
    <Report Type="Photo">
      <Date>2002-12-11</Date>
      <Serial>1101064</Serial>
      <Time>10:10:00</Time>
      <Note></Note>
      <FileName>Daniela_MA_1101064.jpg</FileName>
      <Thumb>TH_Daniela_MA_1101064.jpg</Thumb>
    </Report>
    <Report Type="Photo">
      <Date>2002-12-11</Date>
      <Serial>1101063</Serial>
      <Time>10:09:52</Time>
      <Note></Note>
      <FileName>Daniela_MA_1101063.jpg</FileName>
      <Thumb>TH_Daniela_MA_1101063.jpg</Thumb>
```

```
</Report>
<Report Type="Photo">
  <Date>2002-12-11</Date>
  <Serial>1101062</Serial>
  <Time>10:09:45</Time>
  <Note></Note>
  <FileName>Daniela_MA_1101062.jpg</FileName>
  <Thumb>TH_Daniela_MA_1101062.jpg</Thumb>
</Report>
<Report Type="Photo">
  <Date>2002-12-11</Date>
  <Serial>1101061</Serial>
  <Time>10:09:40</Time>
  <Note></Note>
  <FileName>Daniela_MA_1101061.jpg</FileName>
  <Thumb>TH_Daniela_MA_1101061.jpg</Thumb>
</Report>
<Report Type="Photo">
  <Date>2002-12-11</Date>
  <Serial>1101059</Serial>
  <Time>11:07:58</Time>
  <Note></Note>
  <FileName>Daniela_MA_1101059.jpg</FileName>
  <Thumb>TH_Daniela_MA_1101059.jpg</Thumb>
</Report>
<Report Type="Photo">
  <Date>2002-12-11</Date>
  <Serial>1101058</Serial>
  <Time>10:06:59</Time>
  <Note></Note>
  <FileName>Daniela_MA_1101058.jpg</FileName>
  <Thumb>TH_Daniela_MA_1101058.jpg</Thumb>
```

```
</Report>
<Report Type="Photo">
  <Date>2002-12-11</Date>
  <Serial>1101057</Serial>
  <Time>10:05:26</Time>
  <Note></Note>
  <FileName>Daniela_MA_1101057.jpg</FileName>
  <Thumb>TH_Daniela_MA_1101057.jpg</Thumb>
  </Report><Report Type="Photo">
  <Date>2002-12-11</Date>
  <Serial>1101055</Serial>
  <Time>10:04:04</Time>
  <Note></Note>
  <FileName>Daniela_MA_1101055.jpg</FileName>
  <Thumb>TH_Daniela_MA_1101055.jpg</Thumb>
</Report>
<Report Type="Photo">
  <Date>2002-12-11</Date>
  <Serial>1101054</Serial>
  <Time>10:00:48</Time>
  <Note></Note>
  <FileName>Daniela_MA_1101054.jpg</FileName>
  <Thumb>TH_Daniela_MA_1101054.jpg</Thumb>
</Report>
<Report Type="Photo">
  <Date>2002-12-11</Date>
  <Serial>1101053</Serial>
  <Time>10:00:38</Time>
  <Note></Note>
  <FileName>Daniela_MA_1101053.jpg</FileName>
  <Thumb>TH_Daniela_MA_1101053.jpg</Thumb>
</Report>
```

```
    <Report Type="Photo">
      <Date>2002-12-11</Date>
      <Serial>1101052</Serial>
      <Time>09:52:05</Time>
      <Note></Note>
      <FileName>Daniela_MA_1101052.jpg</FileName>
      <Thumb>TH_Daniela_MA_1101052.jpg</Thumb>
    </Report>
    <Report Type="Photo">
      <Date>2002-12-11</Date>
      <Serial>1101051</Serial>
      <Time>09:51:52</Time>
      <Note></Note>
      <FileName>Daniela_MA_1101051.jpg</FileName>
      <Thumb>TH_Daniela_MA_1101051.jpg</Thumb>
    </Report>
  </Reports>
</Root>
```

Issuing organization	Report number, ISRN	Report type
FOI – Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 Linköping	FOI-R--1277--SE	Technical report
	Research area code 2. Operational Research, Modelling and Simulation	
	Month year May 2004	**Project no.** E7093
	Customers code 5. Commissioned Research	
	Sub area code 21 Modelling and Simulation	

Author/s (editor/s)	Project manager
Dennis Andersson Christer Skagert	Mirko Thorstensson
	Approved by Mirko Thorstensson
	Sponsoring agency Swedish Armed Forces
	Scientifically and technically responsible Mirko Thorstensson

Report title

Managing Massive Datasets from Distributed Tactical Operations

Abstract (not more than 200 words)

Reconstruction and exploration is the foundation of analyzing the complex course of events of distributed tactical operations. As the instruments collecting data from operations become more frequent and more capable of recording multimedia data, the datasets needed to store each operation grow rapidly. This thesis presents tools to manage these large amounts of multimedia data from distributed tactical operations, including modeling the hierarchal structure of the actors in an operation, data collection and exporting collected data to the operation visualization framework MIND, developed at the Swedish Defense Research Agency. This thesis explores two main areas of data management: modeling and accessibility. First, we present a general data model dividing the storage need of a distributed tactical operation into logical blocks. Second, we provide a general object-oriented application programming interface for access to the database. Using these two products we also demonstrate how to build applications for building conceptual models, collecting data and exploring missions.

Keywords

Database, MIND, Distributed Tactical Operation, Data modeling

Further bibliographic information **Language** English

Published master's thesis at the University of Linköping, ISRN LiTH-IDA-EX--04 / 063--SE

ISSN 1650-1942 **Pages** 106 p.

Price acc. to pricelist

Utgivare	Rapportnummer, ISRN	Klassificering
Totalförsvarets Forskningsinstitut - FOI	FOI-R--1277--SE	Teknisk rapport
Ledningssystem	**Forskningsområde**	
Box 1165	2. Operationsanalys, modellering och simulering	
581 11 Linköping	**Månad, år**	**Projektnummer**
	Maj 2004	E7093
	Verksamhetsgren	
	5. Uppdragsfinansierad verksamhet	
	Delområde	
	21 Modellering och simulering	

Författare/redaktör	Projektledare
Dennis Andersson	Mirko Thorstensson
Christer Skagert	**Godkänd av**
	Mirko Thorstensson
	Uppdragsgivare/kundbeteckning
	Försvarsmakten
	Tekniskt och/eller vetenskapligt ansvarig
	Mirko Thorstensson

Rapportens titel (i översättning)

Hantering av stora datamängder från distribuerade taktiska operationer

Sammanfattning (högst 200 ord)

Att återskapa och utforska är en grundläggande metod för att undersöka det komplexa händelseförloppet under en distribuerad taktisk operation. I takt med att det blir allt vanligare att använda datainsamlingsinstrument, och att dessa blir bättre och bättre på att samla in multimedial data, ökar storleken på den mängd data som behöver sparas. Denna rapport lägger fram verktyg för att hantera dessa stora multimedia-databaser, inklusive modellering av den hierarkiska konceptstrukturen inför en övning, datainsamlingen och export av den insamlade datan till visualiseringsramverket MIND som utvecklats av forskare på FOI. Denna rapport utforskar främst två stora delar av datahanteringen, nämligen modellering och åtkomst. Först presenteras en generell datamodell som separerar datan från distribuerade taktiska operationer i logiska block. Sedan visar vi också ett generellt objektorienterat applikationsprogrammeringsgränssnitt (API). Med hjälp av dessa två produkter demonstrerar vi slutligen hur man kan bygga applikationer för att skapa konceptmodeller, samla in data och utforska operationerna.

Nyckelord

Databas, MIND, Distribuerade taktiska operationer, Datamodellering

Övriga bibliografiska uppgifter	Språk Engelska

Publicerad som examensarbete vid Linköpings universitet, ISRN LiTH-IDA-EX--04 / 063--SE

ISSN 1650-1942	Antal sidor: 106 s.
Distribution enligt missiv	Pris: Enligt prislista