CHALMERS

Database for a high performance and stability demanding command and control system

Master of Science Thesis in Electrical Engineering

SOFIA JOHANSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden 2010

MASTER OF SCIENCE THESIS 2010

Database for a high performance and stability demanding command and control system

Master of Science Thesis in Electrical Engineering

SOFIA JOHANSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG

Göteborg, Sweden 2010

Database for a high performance and stability demanding command and control system

SOFIA JOHANSSON

Examiner: DAVID SANDS

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Göteborg
Sweden
Telephone: + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden 2010

Abstract

The purpose of this Master Thesis was to evaluate the possibility to use a database for data storage in a Command and Control system being built by SAAB AB (Saab) for Estonia. Today the data storage consists of a distributed data model and this complicates the wanted functionality.

Data to be stored in the database was analysed and modelled into EER-diagrams. The diagrams were then transformed into database schemas that had to be tested. The database had to be changed during the thesis to get faster. The evaluation tests were performed by using a Java program that created data of the correct format and the data was then used to test the database performance.

The results from this thesis were found to fulfil the system demands when using the test data and test program written. The results will need further evaluation before it is possible to fully understand how including this database would affect the system.

Sammanfattning

Syftet med den här rapporten var att testa möjligheten att använda en databas i ett ledningssystem som byggs av Saab för Estland. Idag består datalagringen av en distribuerad datamodell som komplicerar åtkomsten av de naturliga relationerna som finns mellan data i systemet.

Den data som skulle utgöra databasen analyserades och modellerades till EER-diagram. Diagrammen översattes till ett databasschema som låg till grund för databasen. Databasen testades och ändrades för att bli snabbare. Ett Javaprogram skrevs för att skapa data på rätt format, denna data användes sen för att testa databasen. Även testen utfördes genom ett egenhändigt skrivet testprogram i Java.

Resultaten uppnådda under detta projektarbete ansågs vara tillräckliga för att uppfylla kraven när testdata och testprogram användes. För att fullt ut kunna veta hur systemet skulle påverkas av att ersätta dagens datalagringsmodell med en databas måste fler test göras där tillgång till den verkliga datan och systemet finns.

Acknowledgement

I would like to take the opportunity to thank some people for helping me during the work with this thesis. Sven Nilsson has been my supervisor at Saab and he has been very helpful by giving an introduction to the problem and continually keeping me updated of the system. Josef Svenningsson has been my supervisor at Chalmers and has helped me to approach the problem and structure the work. Jessica Alhbin for her support and encouragement through all the work. The employees at the department for making me feel welcome and let me take part of the daily routine and gossip. I would also like to thank Karolina Ljungberg,Maria Stegberg, Pierre Ingmansson, Judit Sunesson, Marcus Oscarsson and Mattias Runge for valuable feed back, support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traditionally Saab[3] has stored application data in a distributed data structure. In the command and control system covered by this report most application input and output data is related in many ways and different components need access to the same information. The old traditional model makes it hard to find and represent the relationships between data. Therefore it is the wish of Saab to install a relational database using MySQL.

Databases are used to simplify and make the storing and handling of data effective. The idea is to have all data collected in one location so that all the programmers easily know how and where to get hold of the information needed for their application.

When replacing the present data storage model with a database it is important that it does not slow down the data handling process. It must not however jeopardize the stability or persistency of the data storage.

1.1 Aim

The goal of this master thesis is to make an information model and create a database that sufficiently can replace the current data storage application. The database is to be tested on both performance and stability. This is to be done in close cooperation to a real project at Saab, where the new model immediately can be compared to the existing. The database should be able to interact with the existing system.

Chapter 2

Theory

This chapter introduces the theory that this master thesis is based on. It gives a short introduction to the areas of command and control, the type of the system reviewed, database management system which is used to solve the problem and enhanced entity-relationship diagram which was used to visualise the basic idea of this project.

2.1 The system

The system is built for the Estonian (government) military and is called VESP = VSHORADMS Estonia Saab Part where VSHORADMS stands for Very Short Range Air Defence Missile System. The VESP system is a system of systems and consists of 4 parts, the structure of a system can be seen i Figure 2.1

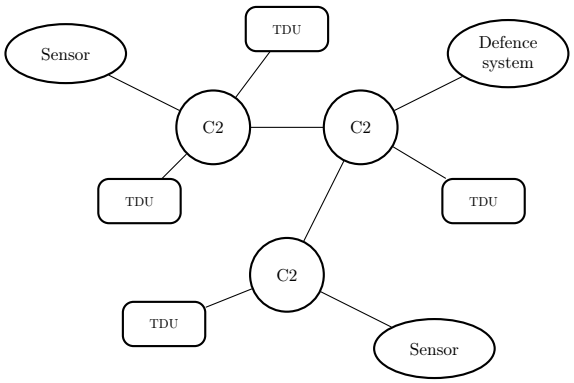

Figure 2.1: VESP

Giraffe AMB (GAMB) The GAMB is a sensor that monitors 360° of air volume and simultaneously detects and warns of incoming ballistic weapons using 3D-radar.[6] The GAMB can be seen in figure 2.2

Figure 2.2: Giraffe

C2 Centre The brain in the system. It is responsible for processing and handling information from the sensor and other C2s. The processed information is then stored locally and can be sent to the other C2s or TDUs.

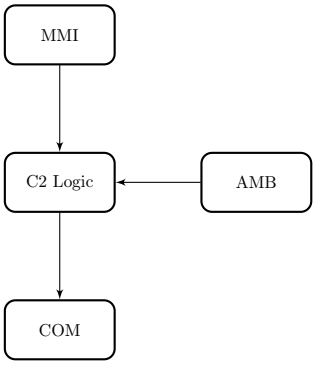

Figure 2.3: An overview of the data flow between internal components in a C2 Center

Tactical Data Unit (TDU) TDU is a portable hand held device that receives data from the C2. The TDU is used by a soldier and together with a weapon they form a fire unit.

Communications (COM) The communications between the different units both internal and external i.e C2-C2 or C2-TDU.

2.1.1 Data flow

Traditionally at Saab the systems consists of an information flow where each component that uses information also processes it and stores it locally in order to be able to use it. An example of this can be seen in Figure 2.4 which shows

the data flow for the construction of a track in the sensor. For this reason there has been no need to use a centralized storage system.

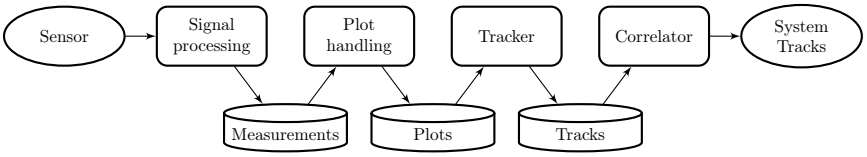

Figure 2.4: Traditionally data flow through data processing unit

In the C2 system there are several different components that need to use and alter the same data. Examples of these components are the Man Machine Interface (MMI) which uses the track data to make a visual representation of the air picture. At the same time the threat evaluation component (TEWA) also needs the track data to determine if a target is a threat. Since a C2 is only one part of the VESP system, there is also need for external communication like the TDU COM which transmits information to the fire units on the battle field, C2 COM that communicates with other C2s and External link which transfers the system information to a external system, e.g the defence system. The correlator uses the track data from different C2s to create a complete air picture that contains information of what all the connected sensors see. The Order Manager uses this information to set and issue orders.

In the traditional system every piece of information is owned by one component and this would mean storing the same information in several places within the C2 system. Therefore the need to evaluate a more centralized approach is clear. The data flow of the C2 system can be seen in Figure 2.5.

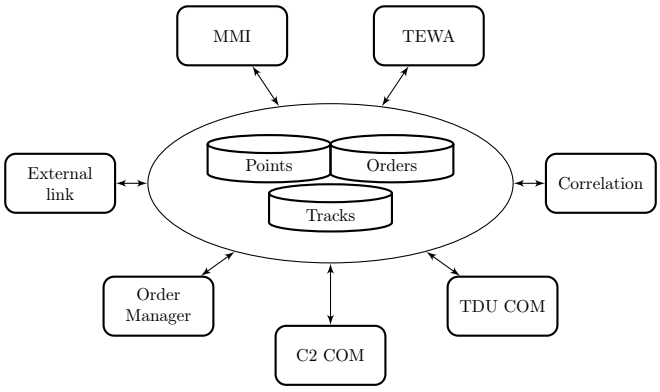

Figure 2.5: Data flow through data processing unit in the VESP system

2.1.2 Command and Control System

A Command and Control System is used to gather information from different sources (i.e. radar, sonar, tactical data links or observers). An administrator evaluates the information and makes a decision on how to act in the specific situation. In military situations the action can be an order to open fire, or in a civilian situation to send paramedics.[9]

The structure of different C2s are hard to find since they are used in military operations and often are classified information. But a general overview of two systems will be described below. The Slovenian army Command and Control system TIPSINK is used to manage the information needed during an operation. It holds information about the state of the army and other friendly or hostile units. The base of the system consists of two main programs IRIS replication mechanism and Sitaware. The first is in charge of the data exchange between the C2s. This information is used for decision-making during operations. The second one is the graphical interface of the C2 system used by an operator to get a general view of the tactical units on the battle field. The graphical interface hold information needed about the situation to be able to make decisions and share it with other military units. [14]

The system which this Master thesis model and the data that is going to be stored in the database is described in the section below. It consists of four major parts.

Missions A mission is defined as a collection of points and tactical settings. It contains information about a single mission and defines the position of the C2 unit, defended assets and fire units. There can be several different missions loaded into the system but only one can be active at once.

Tracks A track is a flying object of interest to be watched. It might be an enemy that needs to be monitored or a friendly air plane. The position and velocity of the track is reported by the Sensor.

Points Points are objects that can be defined on tactical display. There are two types of points, ground points which represent physical objects, or reference points which might be an air defence area or line.

Orders There are two kinds of orders in VESP, Mission Control Orders (MCO) and Fire Control Orders FCO. They can either be local, between a C2 unit and any connected TDU, or remote, between a C2 unit and other reporting units. An order will always have a source and a destination. An FCO connects two or more objects together with an assignment. It might be to assign a fire unit to monitor an airplane of interest.[4]

2.2 Database

There are different kinds of databases to choose between. To mention some there are the relational database, object database, relational-object database and the flat-file database. There are advantages and disadvantages with all of the above mentioned alternatives. The relational database model is the most commonly used database model as of today mainly because it is easy for the programmer to implement and has a good protection from programming errors. It was introduced by Edgar F. Codd in the 1970s. It consists of several tables which represents the data and relationships between it, similar to the mathematical relational theory. To be able to connect to a relational database through an object-oriented programming language e.g. Java an extra interface that translates the data manipulation language to Java and back has to be implemented.

In some cases when there is need of more complex data structures the relational model is inadequate. One case is a multimedia database that must be

able to store and recall video audio, images as well as documents. To meet this demand the object-oriented database was introduced in the 1980s. This database allows the structure to be more general and was therefore thought to be a competitor to the relational database but the complexity and lack of standardisation hindered its success. Object-oriented databases build on the object-oriented programming language rules and include features such as inheritance, object-identity and encapsulation (information hiding). It gives direct access to the database information through the programming language and no additional data manipulation language is needed. This yields low over head when accessing the database but on the other hand it is easier for an error in the programming language to be transferred in to the database.

The object-relational database model is a combination between the relational model and the object-oriented model. It extends the relational model with object orientation, and allows inheritance of relations not just types. An object-relational database is a good option for developers that has a relational database but need to have object-oriented characteristics.

A simpler alternative would be to use a flat file database. A flat file database consists of only one table/file with a field for each of the attributes. The fields are separated by a delimiter for example a comma, the file can then be parsed to receive the wanted information. These files can then be managed by the file processing system and different operational system applications can be built to create more advanced functionality. [12] [15]

2.2.1 Database management system - DBMS

There are a lots of different database management systems to choose between but it was given in the thesis statement that the use of MySQL was desired in this project. MySQL is an open source database widely used all over the world, distributed under a dual license model, where non commercial users can use it for free under the GNU General Public License[7]. Commercial users must buy a license but it is a small cost compared to many other database management systems.

The MySQL database is owned, developed and supported by Sun Microsystems, one of the world's largest contributors to open source software.[1] Sun was recently bought by the big database company Oracle[2]. What effect this might have on MySQL is not yet known.

2.3 Enhanced Entity-Relationship diagram

Enhanced entity-relationship diagram (EER-diagram) is an easy way to communicate the structure and relationships of the data to persons not familiar with the concept of databases. The choice to model the data using the EER-diagram was based on the fact that it is considered to be more efficient than the object-oriented diagram to model the data and easy to use to communicate with other people about the model.[15]

EER is an extended version of ER-modelling that includes specialization/inheritance used to model databases. A description of the different parts of the EER-diagram used in this thesis can be seen in figure 2.6

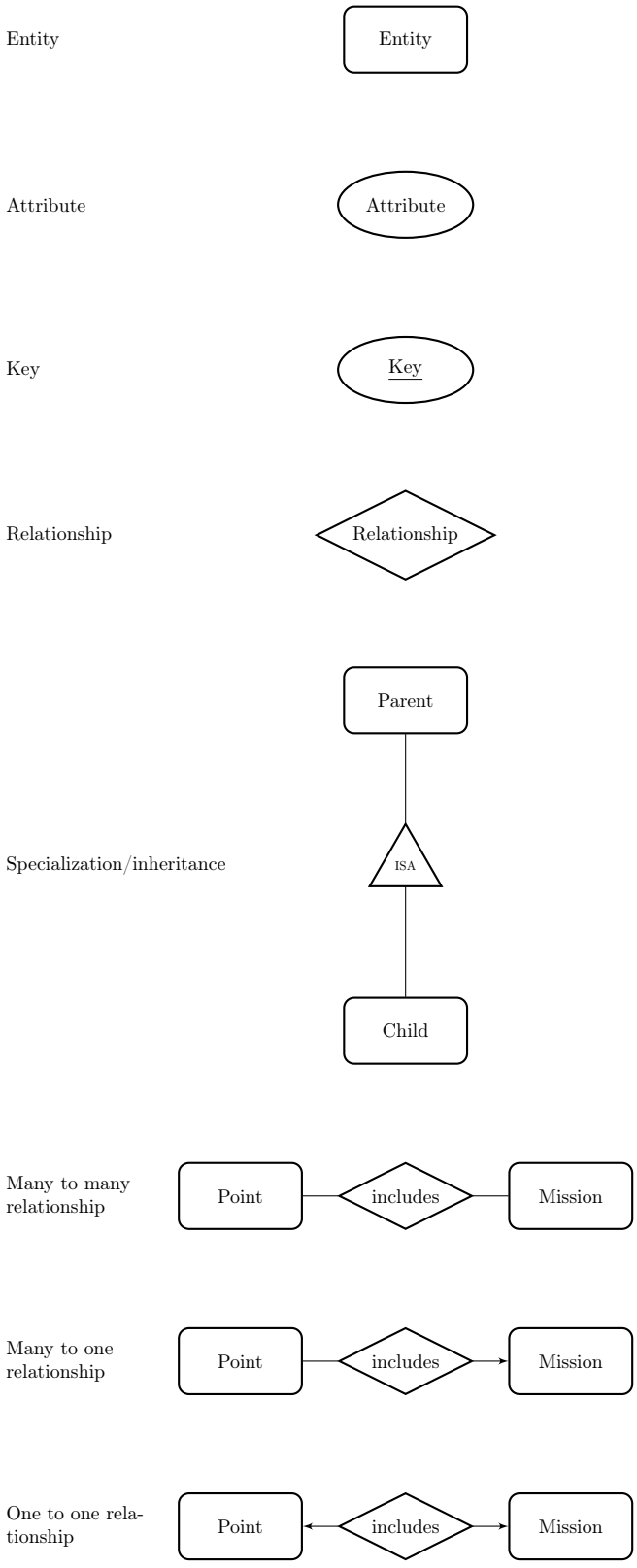

Figure 2.6: EER-parts

2.4 Connecting to the database

The command and control system under construction is written in Java and there has to be a way to make the system talk to the database. This can be done using Java Database Connectivity (JDBC) and the JConnector which is the official JDBC driver for MySQL.

The connection to the database had to be in Java and there was a desire to use Hibernate. There was a concern that Hibernate might slow down the whole process and that could not be risked so JDBC was also evaluated and used.

2.4.1 Java Database Connectivity - JDBC

JDBC is a complementary Application Programming Interface (API) to the Java API used to connect to databases. JDBC is now a part of the Java standard edition which makes the use of databases easier. To be able to connect to the database using JDBC, the JConnector which is the official JDBC driver for MySQL also have to be installed.[8]

2.5 Stored Procedures

Stored procedures are a part of the DBMS used to secure and speed up the database queries. They are stored in the data directory in the same place as the data in the database. Stored procedures are supported in MySQL since MySQL 5.1. Stored procedures are a way to store a collection of statements in the server. This also means that a user can refer to the procedure instead of reinitiate it every time it is used. There are many advantages of using stored procedures e.g. the benefit of uniting clients that use different platforms to only use the same database operations. This increases the security of the database, if the only operations allowed in the database are predecided, the risk of misuse get reduced and the ease of logging increases.

Chapter 3

Method

This chapter describes the methods used in this masters thesis. It explains construction of the database model and database schema. It also describes the tests that have been performed.

3.1 Analysis

Initially information was gathered from books, articles and Saabs documentation. The literature was studied to find out what database modelling option would be most suitable to describe the data and the different relations between them. After studying the gathered information, the key functionality of the system was found to consist of a number of major data types, mainly missions, points, tracks and the different relations between them.

3.2 System

The initial idea of this Master thesis was to implement a database into the C2 center of the system described in section 2.1. This was not possible to carry out since the structure of the data to be stored changed during this thesis and therefore the results of this thesis will just show whether it is possible to carry out this change or not. A test program had to be written and used to test the possibility to integrate a database into the system. The data was modelled and a data model created. The model was later used to realize the database and the database was tested.

3.3 Data model

To model the data, the collected information was analysed and modelled into an EER-diagram as described in section 2.3. The EER-diagram was chosen based on the information given in [10],[13] and the fact that prior knowledge and experience of it gave it clear advantages in the case of less learning time. The data model does not contain information about orders described in 2.1.2 since their structure was not decided in time to be able to include them in this masters thesis. The orders are relations with some extra information and can easily be incorporated into the the real system by adding new relationships.

3.3.1 Missions and Points

A mission is a container of information and there can only be one active mission at a time. A point has to exist within a mission and is persistent data that represent an object. The point is initialized and then only seldom updated. This data type has to be persistent but does not require fast access. A point always consists of at least one position based on if it is representing a point, area or volume. A point may also have a reported position and thus get multiple positions. The created EER-model of missions and points can be seen in Figure 3.1.

A mission is identified by its name in the system. The name is not the best primary key and therefore an auto incremented integer identifier have been used as the primary key. An auto incremented integer has also been used as primary key for points. A point have a unique identifier, a track number, where the different C2s get a separate range of numbers that they are allowed to use for their points. Since a point may have several positions, a position is represented as an entity by itself. A position does not however have a unique identifier, no validations are made to see if a position already exists in the database. The positions also have an auto incremented integer as primary key.

A point can be of many different types and the different types contain a variation of data, therefore points have been divided into the subclasses ground point and fire unit, where fire unit is a subclass of ground point.

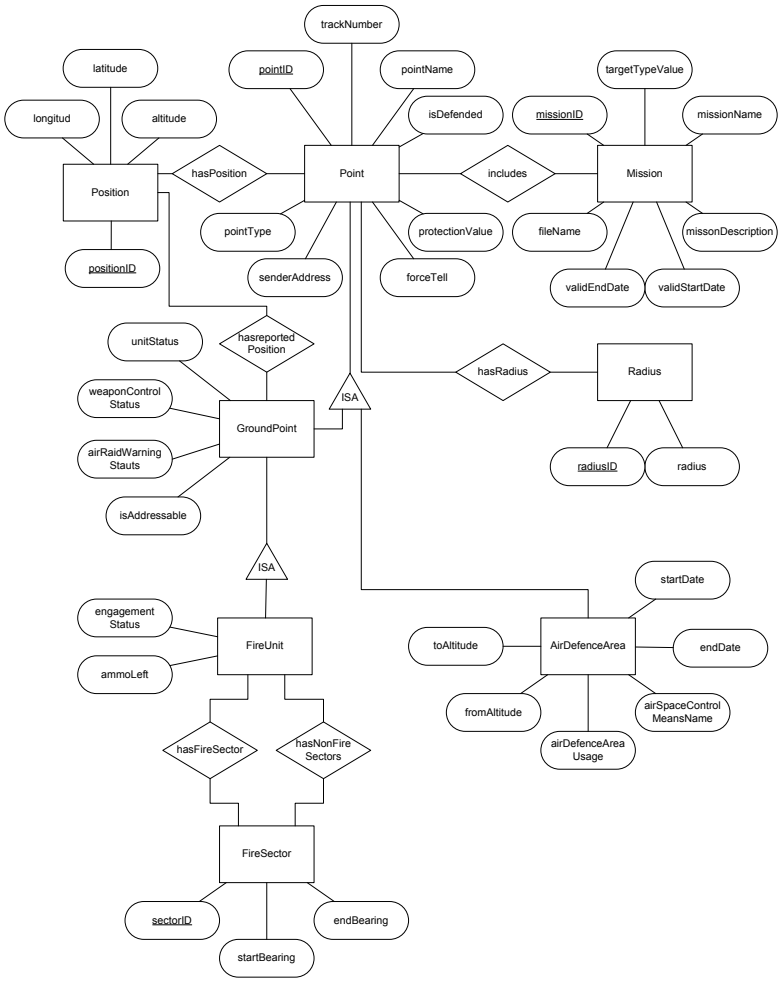

Figure 3.1: EER-diagram of missions and points

3.3.2 Tracks

In this system a track is always an air track and will always have a position and a velocity, and it is this movement that is traced. The position of the track is updated once every second and need to be stored. There can be several hundred tracks in the air at a time and they all need to be stored. The information of the old positions and velocity need not be persisted and may be overwritten. The created model in Figure 3.2 describes the general case of tracks.

A track is identified in the system by its track number. The track number has to be within range of the number assigned to the C2. This could be used as a primary key but to maintain the speed of the system an auto incremented integer have been used. The track may also have more track numbers that

correlate to the other C2s assigned range of number.

This model was created under the impression that the system track might not contain position or velocity. After studying the system the tracks have been found to always be air tracks and always have a position and a velocity therefore the database also had to be altered.

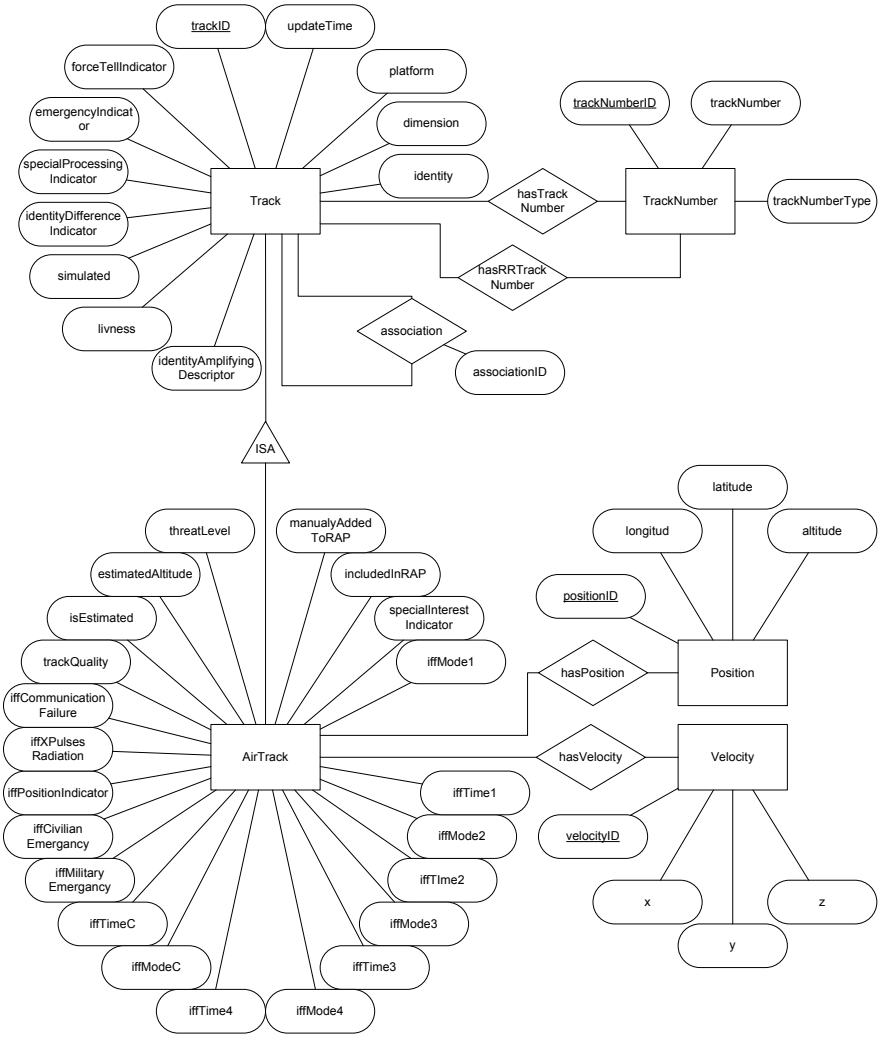

Figure 3.2: EER-diagram of tracks

3.3.3 The complete data model

The models seen in the two previous sections can be combined and then the complete EER-diagram was created and can be seen in Figure 3.3. The system consists of various types of data and associations between them. There are several different types of relations that need to be described. There are associations between tracks and tracks, tracks and points and there is also a relation called orders that has not been modelled since its structure was not defined and therefore not possible to model. Orders associate information to a point or track.

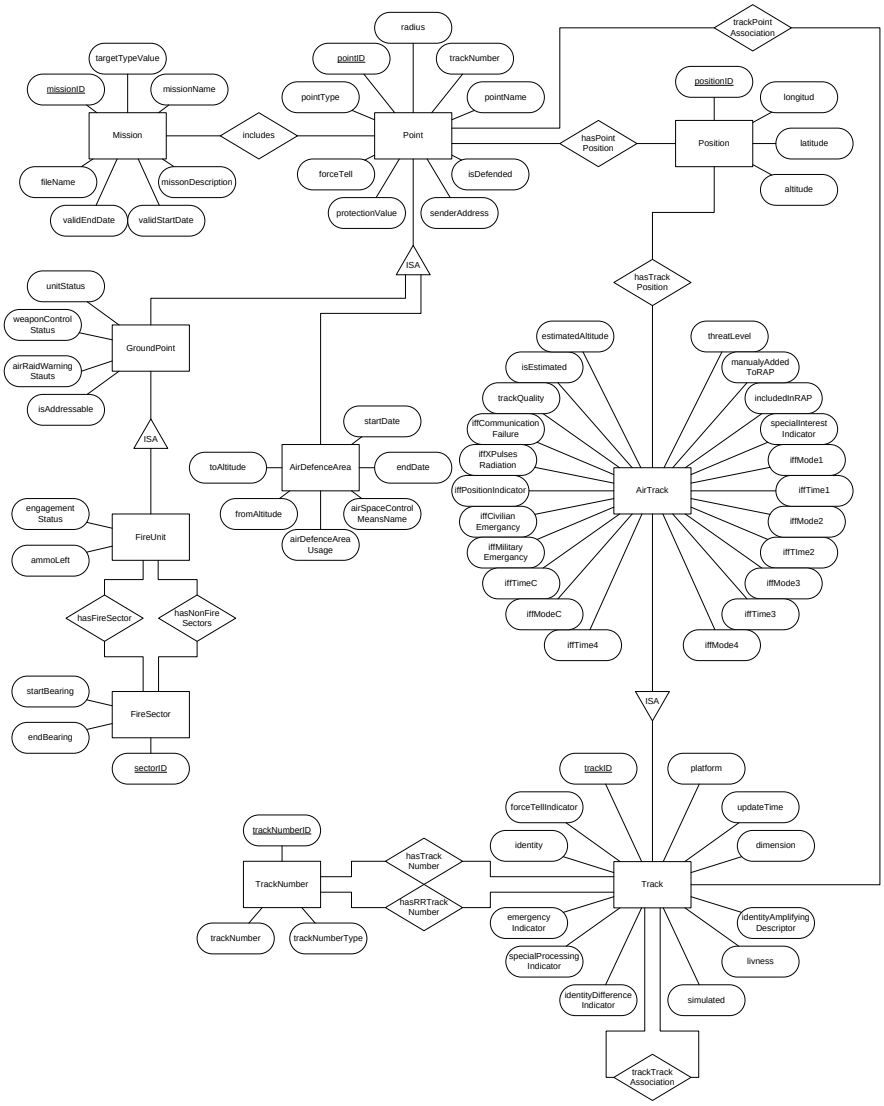

Figure 3.3: EER-diagram

3.4 Database design

The DBMS used in this thesis has been MySQL 5.1 which is the current and recommended version[8]. The database was created through a direct translation of the EER-diagram as described in[11]. When creating the data model, the aim was to make it at least to comply with the third normal form[15]. The design changed during the project due to the fact that the preliminary testing showed that the original design was too slow. The final design can be seen in Appendix C.

In the VESP system the objects have several different identifiers used at different times. In a database one single identifier is to prefer and therefore an auto incremented integer have been used as identifier (primary key) for most of the objects.

There were lots of thoughts on how much redundancy the database can cope with and still be fast enough without causing problems updating the contained information. Some different database models where created to make it possible to test the efficiency of the different models.

3.5 Tests

In this section the different tests and design choices are described. It is also described how the test data is created and how it is used to test the performance of the database. The critical part of the test are going to be the track data. The system is supposed to be able to handle 500 track updates each second. This would mean that there are 500 air planes in the air at the same time that the sensor detects, this is very unlikely but the system still needs to manage it. There are no requirement that the point operations have to be fast but the data stored has to be persistent over time.

3.5.1 Populating the database

Before any testing could be done the database had to be populated. This was accomplished by writing Java code that generated random data on the correct format into a text file that was loaded into the database. A populated database helps make the measurements more correct and make it possible to test the foreign key constraints. The size to which point the tables were filled was based on the information about how much data the system is designed to contain. These generation programs were also used to generate the data used to test the database.

3.5.2 Test data

To be able to run tests on the database data on the correct form needed to be created. This was accomplished by using the previously mentioned Java program to generate random data on the correct format. The data was later used to test the database.

3.5.3 Initial Tests

The initial test was performed on a regular office laptop with encrypted hard drive. The specifications of the computer can be seen in appendix B.1. A

summary of these tests can be seen in Table 4.1. These tests were made to tune the database and modify the database schema. The tests showed that the separation of data into many different tables made the querying too slow to fulfil the system requirement of track speed, and therefore the database had to be adjusted. The improved version was the one used in the final tests. At this point there was a clear advantage of bundling operations together and commit (i.e store) them at once. The bundling were achieved by disabling the auto commit function in the initializing file.

The querying was done by the execution of a file containing SQL statements. i.e. the queries were made directly in the command line in MySQL.

3.5.4 Querying the database

The final test was done by running code on computers similar to the ones going to be used in the real system. The computer specifications can be seen in appendix B.2 The operating system running on these computers are suse linux 10[5]. The tests have been performed several times and the results have been written into a file. The data files have then been studied and the maximum, minimum, mean and median values from every test run have been picked out and inserted into a table which can be seen in Chapter 4.

The test times that are declared in the result chapter are all calculated by the Java-functions and this might affect the collected times.

3.5.5 Stored procedures

Stored procedures were used to increase the speed of the database queries. The queries are preloaded into the database and invoked later when they are going to be used. The use of stored procedures also facilitates the queries by being able to call upon the code already written.

The stored procedures were used through the JDBC connection to the database. The procedures were not created directly into the database but created in the Java code.

Chapter 4

Results

The big question in this project was if the database was going to be able to measure up to the specifications given. The critical point of the results are the track operations that need to be fast. The actual system requirements are classified information and will therefore not be disclosed in this report. Point operations need not to be fast but persistent. The system is supposed to work in close to real-time.

4.1 Initial tests

Initially there was no access to the real system or the computers on which the system was going to run therefore the first tests were made on an office laptop with encrypted hard drive. The results of this test can be seen in summary in Table 4.1. The tests have been performed by bundling of operations and the results are shown for 10, 100 or 1000 operations together at once. The effect of bundling can be seen in the table. The office computer did not fulfil the requirements and as a result of this more tests had to be done using the correct environment.

Table 4.1: Initial Tests

Query	number of objects	time ms/object
Create Track	10	8.55
	100	2.91
	1000	1.93
Create Point	10	7.82
	100	3.19
	1000	2.01
Update Track	10	29.07
	100	24.42
	1000	23.93
Update Point	10	186.88
	100	172.23
Get Track	10	1.31
	100	1.10
Get Point	10	1.93
	100	1.67
Delete Track	10	17.89
	100	6.07
Delete Point	10	11.10
	100	3.31

4.2 Java Tests

Tests have been executed to evaluate the performance of the database using a connection through JDBC. The tables below show the results. There are different tables for all the different tests. The database should be able to handle 500 tracks each second. This means that the execution time of each track operation must not exceed 2ms.

To test the speed of the database queries were asked. The time to create, i.e. store, a track into the database has been measured, by sending different numbers of data to the database at once to see if bundles altered the speed. The tests queries have been executed 1000 times every test run and every test run have been executed five times.

4.2.1 Create

The creation of tracks and points was tested as described in Chapter 3. First the test data had to be created and then the time to store it into the database was measured. The times can be seen in Table 4.2 and Table 4.3. The tables describe the results for 1, 10 or 100 bundled operations respectively and it can be seen that the bundling of operations gives clear advantages in the speed of the querying.

Table 4.2: Create Track

Number of Tracks	Test run	Mean (ms)	Median (ms)	Max (ms)	Min (ms)
1	1	0.9	1	10	0
1	2	0.9	1	9	0
1	3	1.1	1	10	0
1	4	1.3	1	8	0
1	5	1.0	1	10	0
10	1	4.9	4	61	3
10	2	5.0	4	80	3
10	3	4.8	4	66	3
10	4	4.9	4	53	3
10	5	4.9	4	38	3
100	1	38.9	38	460	29
100	2	40.5	38	456	29
100	3	40.2	38	461	29
100	4	38.6	37	463	29
100	5	37.5	37	453	29

Table 4.3: Create Point

Number of Points	Test run	Mean (ms)	Median (ms)	Max (ms)	Min (ms)
1	1	0.9	1	11	0
1	2	1.0	1	12	0
1	3	1.0	1	12	0
1	4	1.0	1	11	0
1	5	1.4	1	12	0
10	1	4.8	4	53	3
10	2	4.9	4	21	3
10	3	4.9	4	119	3
10	4	4.9	4	111	3
100	1	39.7	38	455	29
100	2	42.2	38	423	29
100	3	43.3	38	594	29
100	4	42.1	38	426	29
100	5	40.0	38	478	29

4.2.2 Get

This section contains the results of reading a track or a point from the database. This was done according to the description in chapter 3. The times measured in this test can be seen in Table 4.4 and Table 4.5. The tables describe the results for 1, 10 or 100 bundled operations respectively and it can be seen that the bundling of operations gives advantages in the speed of the querying.

Table 4.4: Get Track

Number of Tracks	Test run	Mean (ms)	Median (ms)	Max (ms)	Min (ms)
1	1	4.4	4	19	4
1	2	4.2	4	8	4
1	3	4.3	4	10	4
1	4	4.4	4	13	4
1	5	4.3	4	7	4
10	1	40.6	41	47	40
10	2	40.8	40	100	40
10	3	40.5	40	54	40
10	4	40.6	40	53	40
10	5	40.6	41	46	40
100	1	401.5	400	433	397
100	2	401.3	400	437	396
100	3	400.6	400	432	396
100	4	401.9	401	431	399
100	5	400.7	400	463	396

Table 4.5: Get Point

Number of Points	Test run	Mean (ms)	Median (ms)	Max (ms)	Min (ms)
1	1	2.1	2	34	1
1	2	2.0	2	12	1
1	3	2.0	2	24	1
1	4	2.0	2	32	1
1	5	2.0	2	19	1
10	1	17.9	18	76	17
10	2	17.9	18	89	17
10	3	17.7	18	57	17
10	4	17.9	18	90	17
10	5	17.9	18	25	17
100	1	170.2	170	255	169
100	2	170.6	170	245	169
100	3	170.1	169	244	168
100	4	169.8	169	212	168
100	5	170.2	170	252	167

4.2.3 Update

The updating of tracks and points was tested as described in Chapter 3. First the test data had to be created and then the time to update it in the database was measured. The times can be seen in Table 4.6 and Table 4.7. The tables describe the results for 1, 10 or 100 bundled operations respectively, worth noting is that the bundling of operations does not give an advantage in the speed of the querying.

Table 4.6: Update Track

Number of Tracks	Test run	Mean (ms)	Median (ms)	Max (ms)	Min (ms)
1	1	0.1	0	1	0
1	2	0.2	0	4	0
1	3	0.2	0	4	0
1	4	0.1	0	4	0
1	5	0.1	0	3	0
10	1	16.9	17	81	16
10	2	16.5	16	22	16
10	3	16.7	16	80	16
10	4	16.6	16	128	16
10	5	16.6	16	38	16
100	1	178.0	177	189	176
100	2	176.1	176	199	174
100	3	176.1	175	329	174
100	4	176.3	176	324	174
100	5	176.0	175	326	169

Table 4.7: Update Point

Number of Points	Test run	Mean (ms)	Median (ms)	Max (ms)	Min (ms)
1	1	137.1	137	142	136
1	2	139.6	140	167	138
1	3	137.5	137	141	137
1	4	138.1	138	148	137
1	5	138.1	138	144	137
10	1	1374.1	1372	1402	1370
10	2	1386.9	1387	1391	1385
10	3	1366.3	1366	1381	1365
10	4	1370.1	1370	1384	1369
10	5	1377.8	1374	1459	1372

Chapter 5

Discussion

The reason for investigating the use of a database in a command and control system was the fact that the data to be stored consists of many different data types with different kinds of relations between them and the wish to have a centralized data structure. Furthermore there are many different computing modules that use the same data for calculations. The stored data has to stay consistent even though different calculations are performed on it. Functions and computing modules has to be able to be changed or added without any changes to the data. The adding and removing of functionality must be easy. The wish was to create a modular system which is easy to expand and develop new functionality on. A database makes the storing of different kinds of data in one place, add on of new functions and removal of old ones easy. A database comes with a lot of functionality that might not all be needed in this specific case but it is still a good alternative for storing data. One reason not to use a database in this specific case could be if it severely slows down the data processing. In this case the speed of the system is of highest importance, since it represents moving objects in near real time.

5.1 Analysis

Due to the fact that the system description kept changing during the thesis work it was difficult to perform the analysis of the system structure. This meant that the system description had to be reviewed and the result remodelled several times. To be able to perform the desired tests within the time of the thesis work a decision was made to model the data as it was described at the time. This had the effect that the resulting data model does not represent the data structure as it looks in the final version of the VESP system. This means that an implementation of the database concept into the system has to be remodelled because not all relations and data are consistent with the current model.

5.2 Data model and database design

The EER-diagram was chosen as a way to model the information collected in the analysis phase. The EER model was found to be a good way to visualize the data and connections between them. It was also a good base to start the database

design from. The first database schema was a direct translation of the EER-diagram seen in Figure 3.3. The testing showed that the data model was too slow to fulfil the demands on the system due to the fact that there were too many tables to describe the data. The more tables that have to be searched to find or store an object, the slower it became. To enhance the speed of the database some of the tables were combined into larger tables. This meant that a search or store operation had to make fewer searches and therefore became faster. There have not been any changes to the data model after it was established that a database created directly from the model was too slow. Although operations using the point object does not require the same speed as for tracks they were also merged into fewer tables to get consistency in the measurements. This might not have been necessary and is therefore one of the things that need to be taken under consideration in a future implementation.

The results from Chapter 4 show that there are some questions whether the structure used in this thesis is fast enough or not. Therefore other options might have to be taken under consideration. One option could be to store the data into a file but this makes the access to the data and relations between data difficult to manage. A regular file is hard to index and search, relations between different data objects has to be made by adding the extra information into the file and the specific object related to that information. This makes the system data redundant and may cause the system to have more than one copy of the same information and occupy more disk space. An update to only one of the copies causes the system to contain different versions of the same data and this could mean that an invalid data value can be wrongly used. To prevent the creation of corrupt duplicate data, the application has to have a method to check that every copy of the same data gets updated, this might cause a lot of extra workload and delays for the application. This also aggravates the development, change and expansion of the system. Expansions are made more difficult since an addition of a new relation in a file system has to be made either by adding a new file containing all the related information or by indexing the files so that the relation only uses the index to the data instead of duplicating the data, this would practically be the same as creating a simple database without the DBMS.

Another option might be to store the data in some kind of program language related alternative. In this case that would mean in the Java application. This is similar to what is implemented today. Some of the data in this system has to be persistent and since a Java object loses its memory when the application is turned off it would have to be complemented with a file. The advantage of this kind of storage is that it is fast but the disadvantage is that the access from many different components are hard to manage and the system gets hard to evolve without having to change the structure.

There might be a possibility to use a combination of a Java solution and a storing to file or a database to increase the speed of the system. Some of the data must be updated fast and does not necessarily have to be persistent. Perhaps this data can be a Java object and the persistent data can be stored in a file or a database. There is still the problem of relating the different data, this has to be managed by the Java application and to investigate the possibilities to use a system of this kind further evaluation has to be done.

The best option in my opinion would be to optimize the database code so that the data processing gets fast enough. The functions included in the database may not be used at this time but might be needed further along. Not

all functions that could be used at this time are included and tested in the thesis but has to be evaluated later on if Saab wants to use them in their system.

When using a DBMS like MySQL there are many different storage engines to choose between and they all have different advantages and disadvantages. The storage engine can be chosen by looking at what properties and functionalities the application is in need of. In many cases a combination of storage engines are possible to use but the speed of the access to the database tables may differ according to the storage engine type. To increase the speed of data collection from the database one option could be to use a storage engine that only stores the data in the memory. The memory storage engine makes the look up in the tables faster but no entries in the table will be saved if the system goes down or is turned off. The table structure on the other hand will be saved. This could be suitable for the track data since this data does not need to be persistent or accessible after a shutdown. One of the disadvantages with using a memory table is that it does not support foreign keys that was one of desires when creating the database.[8] The InnoDB storage engine used in the thesis work is the only one that supports foreign keys and transactions as asked for in the specifications. All the tests in the thesis have been performed using the InnoDB storage engine.

5.3 Populating the database and creation of test data

The database was populated to better reflect the real system. A populated database gives a better measurement of the result times for operations in the actual system since it is known that the system contains information when in use. The test data was created using a small self written Java program that created correctly formatted random data to be stored in the database. The same program also created the data used to test the different operations, which gave the results seen in Chapter 4. Data used in these tests are random and may thus differ from the ones created in the real system.

5.4 Tests

Test times were calculated by the self written Java program used to make the database queries. This might have effected the resulting times and has to be taken in to consideration when evaluating the results.

Initially there were no access to the computers dedicated to develop the real system on, therefore the first tests had to be performed on a office laptop with encrypted hard drive. The encrypted hard drive most likely effected both the storing and retrieval of data. The size of this effect was hard to estimate and has therefore not been declared for. The results from these tests did not measure up to the requirements of the system but they were used to tweak and optimize the database. The results from this initial test was unfortunately lost in a computer crash and will therefore not be declared in this report. At the end of the thesis work the test environment was accessible and the optimized code could be tested on the VESP system computers.

the system demands. This meant that the data model had to be rearranged to make the database faster. The new model however are more compact and

might not separate the data enough.

As can be seen in Chapter 4, there are big differences in times between the different operations. This could be explained by the complexity of the queries. The speed of the create operations can be traced to the auto incremented identifiers and that no search is required before an insertion. It can also be noticed that bundling of operations increases the speed of operations. To collect information from the database one must find the right data to retrieve and therefore a search has to be done. The search can be made easier if the database is sorted in some way but a sorted database also means that the insertion times increases since a search has to be done here instead. It can be seen from the result table that the laptop test in this case was faster than the one performed in the correct environment. This is a result of the fact that the laptop test only collects informations from one single table using the identifier for that table while the final test use multi table search. One of the disadvantages of using many tables and complex identifiers can be seen when comparing the easy queries on the slow laptop with the complex queries on the faster computers in the laboratory.

Chapter 6

Conclusion

When looking at the model of the data to be stored in the database and the different relations between it there are some major advantages of implementing a database instead of the current distributed data storage model. This thesis have investigated if a database could be integrated into the VESP system and replace the current data storage model and still fulfil the system requirements.

The critical point in the measurements were the speed of the track operations. After evaluating the results of the track operations the mean of the achieved times were found to fulfil the requirements. There were some peek times that only occurred one time out of a thousand that might jeopardize the overall result. This means that the database would fulfil the requirements but additional test needs to be done to assure the correctness of this result with the use of the real data.

There were no requirements that the point operations had to be fast, therefore the times achieved in update point operation will not affect the conclusion that a database could replace the current data storage model. More tests need to be performed to optimize the point operations and retest them.

Chapter 7

Future Outlook

The wider idea to actually integrate the database into to the system was a task that the time available for this master thesis did not allow. To fully understand the possibility to integrate a database into the VESP system, the database has to be modelled to fit the present specifications of what data is going to be stored. Tests must be done in the application where the correct data is used and sent through the real system. Integration of a database will require changes in the existing system. The structure of the data handling needs to be evaluated again. As orders has not been covered by the thesis work they have to be included in a future implementation.

Some recommendations are to test the different table types (Storage engines) of MySQL and see if there is a distinct difference in speed. The different storage engines have different advantages and the needs of the system has to be taken into consideration and the storage engine reevaluated. It would also be of interest to look into how concurrent access of the database would affect the performance. There are lot of different components that need access to the information to be stored in the database.

A system that needs to be run in near real time is affected by the speed of the computer and therefore will keep getting faster. Because of this the results will become inaccurate quickly.

Bibliography

[1] Mysql. `www.mysql.com`.

[2] Oracle. `www.oracle.com`.

[3] Saab microwave system ab. `www.saabgoup.com`.

[4] Saab microwave system internal documentation. Internal.

[5] Suse linux. `www.novell.com/linux/`.

[6] Giraffe amb. `http://www.saabgroup.com/en/ProductsServices/products_az.htm`, Sep 2009.

[7] Gnu general public license. `http://www.gnu.org/copyleft/gpl.html`, Sep 2009.

[8] Mysql. `http://www.mysql.com/`, Sep 2009.

[9] Wikipedia. `http://en.wikipedia.org/wiki/Command_and_control`, Sep 2009.

[10] Pedersen A. Database design resource. `http://www.databasedesign-resource.com/normal-forms.html`, Apr 2008.

[11] Silberschatz A, Korth H, and Sudershan S. *Database system concepts*. McGraw-Hill Companies, Inc., fourth edition, 2002.

[12] Silberschatz A, Korth H, and Sudershan S. *Database system concepts*. McGraw-Hill Companies, Inc., fifth edition, 2006.

[13] Garcia-Molina H, Ullman J, and Widom J. *Database system the complete book*. Pearson Education Inc., 2002.

[14] Joze M, Fras M, and Cucej Z. New approach to the modeling of command and control information system. *Military Communications Conference, 2008. MILCOM 2008. IEEE*, 2008.

[15] Elmasri R and Navathe S. *Fundamentals of database system*. Pearson Education, Inc., fourth edition, 2004.

Appendices

A Abbreviations

Abbreviations in alphabetical order

AMB Agile Multi-Beam

API Application Programming Interface

C2 Command and Control

DBMS Database Management System

EER-diagram Enhanced entity-relationship diagram

FCO Fire Control Orders

GAMB Giraffe AMB

JDBC Java Database Connectivity

MCO Mission Control Orders

MMI Man-machine interface

MySQL My Structured Query Language

ODBC Open Database Connectivity

SMW Saab Microwave Systems

SQL Structured Query Language

TDN Tactical Data Network

TDU Tactical Data Unit

TDUN Tactical Data Unit Network

TEWA Threat Evaluation and Weapon Allocation

VESP VSHORADMS Estonian Saab Part

VSHORADMS Very Short Range Air Defence Missile System

B Computer Specifications

B.1 Latitude D600

- Latitude D600 Pentium M 1.4GHz +14.1IN
- 1.0GB 266MHz DDRAM memory, (2X512MB)
- 40GB (5,400RPM) 9.5mm ide hard drive
- Operating system: Windows XP

B.2 Optiplex 745 MT

- Optiplex 745 Mt - core 2 duo E6300 1.86GHz
- Memory: 2048MB (2X1024MB) 667MHz DDR2 D
- Hard drive: 80GB (7200RPM) 3.5IN Serial
- Operating system: SUSE Linux Enterprise 10.1

C Database Code

Listing 1: Create

```
CREATE TABLE Missions (
        mission_id BIGINT NOT NULL,
        mission_name VARCHAR(10) NOT NULL,
        mission_description VARCHAR(256) NOT NULL,
        valid_start_date BIGINT NOT NULL,
        valid_end_date BIGINT NOT NULL,
        file_name VARCHAR(10) NOT NULL,
        target_type ENUM('No_Statement',
                'Missile_Carrier',
                'Remotely_Piloted_Vehicle',
                'Helicopter',
                'Miscellaneous_Fixed_Wing',
                'Reset_To_No_Statement'),
        target_value SMALLINT NOT NULL
);

CREATE TABLE Points (
        point_id BIGINT NOT NULL,
        point_type ENUM('C2Unit',
                'Sensor',
                'Mistral',
                'Anti_Aircraft_Artillery',
                'VIS', 'Supporting_Unit',
                'SAM',
                'Air_Base',
                'Military_Head_Quarter',
                'Enemy',
                'Enemy_SAM',
                'Enemy_Air_Base',
                'Tactical_Data_Link_Site',
                'Air_Def_Area',
                'Ref_Area',
                'Ref_Line',
                'Ref_circle',
                'Ref_single_Point'),
        point_name VARCHAR(10) NOT NULL,
        is_defended BOOLEAN NOT NULL,
        protection_value INT NOT NULL,
        force_tell BOOLEAN NOT NULL,
        radius DOUBLE DEFAULT 0,
        unit_status ENUM('Available',
                'Not_Available'),
        weapon_control_status ENUM('Weapons_Free',
                'Weapons_Tight',
                'Weapons_Hold',
                'No_Status_Statement')
                DEFAULT 'No_Status_Statement',
        air_raid_warning_status ENUM('White',
                'Yellow',
                'Red',
                'No_Warning_Statement')
                DEFAULT 'No_Warning_Statement',
        is_addressable BOOLEAN NOT NULL DEFAULT false,
        engagement_status ENUM('Not_Fire_Unit',
                'Engage',
                'Cover',
                'Weapon_Assigned',
```

```
                'Covering',
                'Tracking',
                'Fiering',
                'Effective',
                'Not_Effective',
                'Engagement_Broken')
                DEFAULT 'Not_Fire_Unit',
        ammo_left SMALLINT NOT NULL DEFAULT 0,
        fire_unit BOOLEAN DEFAULT false,
        start_date BIGINT NOT NULL DEFAULT 0,
        end_date BIGINT NOT NULL DEFAULT 0,
        air_space_control_means_name VARCHAR(10)
                NOT NULL DEFAULT 'Not_ADA',
        air_defence_area_usage VARCHAR(256)
                NOT NULL DEFAULT 'Not_ADA',
        to_altitude DOUBLE NOT NULL DEFAULT 0
);

CREATE TABLE Fire_sectors (
        sector_id BIGINT NOT NULL,
        start_bearing REAL NOT NULL,
        end_bearing REAL NOT NULL
);

CREATE TABLE Tracks (
        track_id BIGINT NOT NULL,
        update_time BIGINT NOT NULL,
        platform ENUM('No_Statement',
                'Missile_Carrier',
                'Remotely_Piloted_Vehicle',
                'Helicopter',
                'Miscellaneous_Fixed_Wing',
                'Reset_To_No_Statement'),
        dimension ENUM('Dimension_1D',
                'Dimension_2D',
                'Dimension_3D'),
        identity ENUM('Pending',
                'Unknown',
                'Assumed_Friend',
                'Friend',
                'Neutral',
                'Suspect',
                'Hostile'),
        identity_amplifying_descriptor ENUM('None',
                'Faker',
                'Joker'),
        liveness ENUM('Alive',
                'Dying',
                'Dead'),
        simulated BOOLEAN NOT NULL,
        identity_difference_indicator BOOLEAN NOT NULL,
        special_processing_indicator BOOLEAN NOT NULL,
        emergancy_indicator BOOLEAN NOT NULL,
        force_tell_indicator BOOLEAN NOT NULL,
        threat_level ENUM('No_Threat',
                'Low',
                'Medium',
                'High'),
        manually_added_to_rap BOOLEAN NOT NULL,
        included_in_rap BOOLEAN NOT NULL,
        special_interest_indicator BOOLEAN NOT NULL,
        iff_mode_1 SMALLINT,
        iff_time_1 BIGINT,
        iff_mode_2 SMALLINT,
```

```
        iff_time_2 BIGINT,
        iff_mode_3 SMALLINT,
        iff_time_3 BIGINT,
        iff_mode_4 SMALLINT,
        iff_time_4 BIGINT,
        iff_mode_c SMALLINT,
        iff_time_c BIGINT,
        iff_military_emergancy BOOLEAN NOT NULL,
        iff_civilian_emergancy BOOLEAN NOT NULL,
        iff_position_indicator BOOLEAN NOT NULL,
        iff_x_pulse_radiation BOOLEAN NOT NULL,
        iff_communication_failure BOOLEAN NOT NULL,
        track_quality SMALLINT,
        is_estimated BOOLEAN NOT NULL,
        estimated_altitude REAL,
        x DOUBLE NOT NULL,
        y DOUBLE NOT NULL,
        z DOUBLE NOT NULL
);


CREATE TABLE Track_numbers (
        track_number_id BIGINT NOT NULL,
        track_number VARCHAR(5) NOT NULL,
        track_number_type ENUM('Sensor',
                'Tdn',
                'Link')
);

CREATE TABLE Positions (
        position_id BIGINT NOT NULL,
        longitude DOUBLE NOT NULL,
        latitude DOUBLE NOT NULL,
        altitude DOUBLE NOT NULL
);

CREATE TABLE include (
        mission BIGINT NOT NULL,
        point BIGINT NOT NULL
);

CREATE TABLE has_point_position (
        point BIGINT NOT NULL,
        position BIGINT NOT NULL,
        reported BOOLEAN NOT NULL
);

CREATE TABLE has_track_position (
        track BIGINT NOT NULL,
        position BIGINT NOT NULL
);

CREATE TABLE has_fire_sector (
        point BIGINT NOT NULL,
        sector BIGINT NOT NULL,
        non_fire_sector BOOLEAN
);

CREATE TABLE track_has_track_number (
        track BIGINT NOT NULL,
        track_number BIGINT NOT NULL,
        rr_tracknumber BOOLEAN
);
```

```
CREATE TABLE point_has_track_number (
        point BIGINT NOT NULL,
        track_number BIGINT NOT NULL
);

CREATE TABLE track_track_association (
        track_1 BIGINT NOT NULL,
        track_2 BIGINT NOT NULL,
        ass VARCHAR(64)
);

CREATE TABLE track_point_association (
        track BIGINT NOT NULL,
        point BIGINT NOT NULL,
        ass VARCHAR(64)
);
```

Listing 2: Primary Keys

```
ALTER TABLE Missions ADD CONSTRAINT PK_mission
        PRIMARY KEY (mission_id);

ALTER TABLE Missions MODIFY COLUMN mission_id
        BIGINT NOT NULL DEFAULT NULL AUTO_INCREMENT;

ALTER TABLE Points ADD CONSTRAINT PK_point
        PRIMARY KEY AUTO_INCREMENT(point_id);

ALTER TABLE Points MODIFY COLUMN point_id
        BIGINT NOT NULL DEFAULT NULL AUTO_INCREMENT;

ALTER TABLE Fire_sectors ADD CONSTRAINT PK_fire_sector
        PRIMARY KEY AUTO_INCREMENT(sector_id);

ALTER TABLE Fire_sectors MODIFY COLUMN sector_id
        BIGINT NOT NULL DEFAULT NULL AUTO_INCREMENT;

ALTER TABLE Tracks ADD CONSTRAINT PK_track
        PRIMARY KEY AUTO_INCREMENT(track_id);

ALTER TABLE Tracks MODIFY COLUMN track_id BIGINT NOT NULL
        DEFAULT NULL AUTO_INCREMENT;

ALTER TABLE Track_numbers ADD CONSTRAINT PK_track_number
        PRIMARY KEY AUTO_INCREMENT(track_number_id);

ALTER TABLE Track_numbers ADD CONSTRAINT UNIQUE_track_number
        UNIQUE INDEX(track_number, track_number_type);

ALTER TABLE Track_numbers MODIFY COLUMN track_number_id
        BIGINT NOT NULL DEFAULT NULL AUTO_INCREMENT;

ALTER TABLE Positions ADD CONSTRAINT PK_position
        PRIMARY KEY AUTO_INCREMENT(position_id);

ALTER TABLE Positions MODIFY COLUMN position_id
        BIGINT NOT NULL DEFAULT NULL AUTO_INCREMENT;

ALTER TABLE include ADD CONSTRAINT PK_include
        PRIMARY KEY(mission, point);

ALTER TABLE has_point_position
        ADD CONSTRAINT PK_has_point_position
        PRIMARY KEY (point, position);

ALTER TABLE has_track_position
        ADD CONSTRAINT PK_has_track_position
        PRIMARY KEY (track, position);

ALTER TABLE has_fire_sector
        ADD CONSTRAINT PK_has_fire_sector
        PRIMARY KEY (point, sector);

ALTER TABLE track_has_track_number
        ADD CONSTRAINT PK_has_track_number
        PRIMARY KEY (track, track_number);

ALTER TABLE track_track_association
        ADD CONSTRAINT PK_track_track_ass
        PRIMARY KEY (track_1, track_2);
```

```
ALTER TABLE track_point_association
        ADD CONSTRAINT PK_track_point_ass
        PRIMARY KEY (track, point);
```

Listing 3: Foreign Keys

```
ALTER TABLE include ADD CONSTRAINT FK_mission_with_point
        FOREIGN KEY (mission) REFERENCES Missions (mission_id)
        ON DELETE CASCADE;

ALTER TABLE include ADD CONSTRAINT FK_point_in_mission
        FOREIGN KEY (point) REFERENCES Points (point_id)
        ON DELETE CASCADE;

ALTER TABLE has_point_position ADD CONSTRAINT
    FK_point_with_position
        FOREIGN KEY (point) REFERENCES Points (point_id)
        ON DELETE CASCADE;

ALTER TABLE has_point_position ADD CONSTRAINT FK_position_of_point
        FOREIGN KEY (position) REFERENCES Positions (position_id)
        ON DELETE CASCADE;

ALTER TABLE has_track_position ADD CONSTRAINT
    FK_track_with_position
        FOREIGN KEY (track) REFERENCES Tracks (track_id)
        ON DELETE CASCADE;

ALTER TABLE has_track_position ADD CONSTRAINT FK_position_of_track
        FOREIGN KEY (position) REFERENCES Positions (position_id)
        ON DELETE CASCADE;

ALTER TABLE has_fire_sector ADD CONSTRAINT FK_has_fire_sector
        FOREIGN KEY (point) REFERENCES Points (point_id)
        ON DELETE CASCADE;


ALTER TABLE has_fire_sector ADD CONSTRAINT FK_fire_sector_of_point
        FOREIGN KEY (sector) REFERENCES Fire_sectors (sector_id)
        ON DELETE CASCADE;

ALTER TABLE track_has_track_number ADD CONSTRAINT
    FK_track_with_track_number
        FOREIGN KEY (track) REFERENCES Tracks (track_id)
        ON DELETE CASCADE;

ALTER TABLE track_has_track_number ADD CONSTRAINT
    FK_track_number_of_track
        FOREIGN KEY (track_number) REFERENCES Track_numbers
            (track_number_id)
        ON DELETE CASCADE;

ALTER TABLE track_track_association ADD CONSTRAINT
    FK_track_associated
        FOREIGN KEY (track_1) REFERENCES Tracks (track_id)
        ON DELETE CASCADE;

ALTER TABLE track_track_association ADD CONSTRAINT
    FK_track_associated_to_track
        FOREIGN KEY (track_2) REFERENCES Tracks (track_id)
        ON DELETE CASCADE;

ALTER TABLE track_point_association ADD CONSTRAINT
    FK_track_associated_to_point
        FOREIGN KEY (track) REFERENCES Tracks (track_id)
        ON DELETE CASCADE;
```

```
ALTER TABLE track_point_association ADD CONSTRAINT
    FK_point_associated_to_track
        FOREIGN KEY (point) REFERENCES Points (point_id)
        ON DELETE CASCADE;
```