ATC Airport Builder

- **A shapefile tool for simulation purposes.**

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg
Computer Science

Bachelor thesis:
Anna Di Julio
Mikael Karlsson

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
E-husets tryckeri
Lunds universitet
Lund 2013

Abstract

Saab supplies the global market with products, services and solutions ranging from military defense to civil security. One of their many offices is located in Helsingborg where a new product for training air traffic controllers (Saab ATC training solution) is developed. This product offers air traffic control students a realistic 3d environment where they may go through a variety of supervised air traffic control scenarios. In order to stay ahead of competitors, Saab strives to continuously improve their products. Today, it is possible to add airports to the Saab ATC training solution; however, this is a tedious process. Another area of improvement is the way in which map information is utilised in the simulator. Currently, the simulator knows very little about airport environments. By utilising existing map data, the simulator can make more intelligent decisions inside airport areas. This thesis describes how map information in existing shapefiles can be used to enhance airport simulation. In addition, this thesis, describe how the existing 3d engine can benefit from shapefile data.

A prototype, ATC AirportBuilder, has been created to prepare map information for further use by the Saab ATC training solution. ATC AirportBuilder imports shape file data, adapts and exports it to a database. An analysis has also been conducted where several integration possibilities with the simulator and the 3d engine have been evaluated.

Keywords: Shapefile, GIS, Air Traffic Control, SharpMap, GDAL, Simulator

Sammanfattning

Saab förser den globala marknaden med världsledande lösningar, produkter och tjänster som sträcker sig från militärt försvar till civil säkerhet. Ett av Saabs många kontor ligger i Helsingborg, där en ny produkt för träning av flygledare (Saab ATC training solution) utvecklas. Produkten erbjuder flygledarelever en verklighetstrogen 3d-miljö i vilken de kan genomgå olika kontrollerade flygledarscenarion. För att ligga steget före konkurrenter vill Saab kontinuerligt förbättra sina produkter. I nuläget är det möjligt att lägga till nya flygplatser till Saab ATC training solution men det är en tidskrävande process. Ett annat förbättringsområde är hur kartinformation utnyttjas i simulatorn. Simulatorn vet i dagsläget väldigt lite om flygplatsmiljön. Genom att utnyttja befintlig kartdata skulle simulatorn kunna ta mer intelligenta beslut inom flygplatsområdet. Detta examensarbete undersöker hur kartinformation i redan existerande shapefiler kan utnyttjas för att förbättra flygplatssimulering. Därförutom beskrivs i detta examensarbete hur den befintliga 3d motorn kan utnyttja shapefile data.

En prototyp, ATC AirportBuilder, har skapats för att förbereda kartinformationen för att användas vidare i ATC training solution. ATC AirportBuilder importerar shapefile data, anpassar samt exporterar den till en databas. En analys har även genomförts där olika integrationsmöjligheter med simulator och 3d-engine utretts.

Nyckelord: Shapefile, GIS, Air Traffic Control, SharpMap, GDAL, Simulator

Foreword

This thesis project was conducted by two students at Lund University, LTH School of Engineering. The thesis and associated prototype are the result of a project initiated by Saab in Helsingborg.

We would like to thank our examiner at LTH, Christin Lindholm for continuous support. The employees at Saab deserve our thanks for all questions answered, doors opened, and encouraging conversations. Foremost, we would like to thank our main advisor, Mathias Lubera, whose ideas have contributed greatly to this thesis. In addition, his helpful discussions and guidance throughout the entire thesis project has enhanced the quality of this work.

Finally, we would like to thank Victor Sandberg and Daniel Thunberg for many interesting conversations on related topics.

List of contents

1 Background

Saab supplies the global market with a multitude of different world-leading solutions, products and services, ranging from military defense to civil security. In order to stay sharp Saab constantly develops, adopts and improves new technology to meet customers' changing needs on all continents.[1]

In this thesis the development of a prototype made to improve such technology is discussed and evaluated. Saab develops management systems for training and simulation. Within this field Saab has recently launched a new product for training of air traffic controllers. (Saab ATC solution). This system consists of several different parts. Two out of which are a simulator (ATSsim)where scenarios can be created and a 3d engine providing the students with 3d environments of airports. A customer need that has been brought into light is the possibility to add new airports to their system. This can be done today but the process is cumbersome as it requires manual modelling of airport data to create corresponding 3d environments.[2]

This is where this thesis fits in as it discusses a method in which the modeling of airport data can be simplified. A prototype is made for viewing and modifying airport data in a simple manner, preparing this data for possible future automatic 3d mapping.

1.1 Air traffic control

Air traffic control (ATC), a service providing directions to aircrafts on the ground or through controlled airspace. The main purpose of this service, provided by ground-based controllers is to keep distances between aircrafts in the system. Apart from collision avoidance, the ATC system is to make more efficient use of airspace and provide additional service to pilots. Such services may be navigational assistance or weather avoidance.

The ATC services provided may vary in different countries, where some provide no ATC services at all. Definitions for different airspace operations and classes of which aircrafts may operate have been defined by The

[1] SAAB AB, 2013. Saab in brief, http://www.saabgroup.com/en/About-Saab/Company-profile/Saab-in-brief/(Accessed 2013-05-21).
[2] Lubera, Mathias; Software Engineer, Saab Training and Simulation. WISE connectivity, lecture, 2013-04-04.

International Civil Aviation Organization (ICAO). These very specific guidelines help unite differences between different nations.[3]

1.2 Saab, technology & systems

Saab (Svenska Aeroplan AB) was founded in 1937 in the city of Trollhättan where the production of flight military units starts. In 1944 Saab expanded its marketing targets with civil aircrafts and two years later (1946) the first Saab car project saw daylight. In the 1950th as aircrafts became continuously more complex in terms of their systems. Saab's focus lay now not only on airframes but also on control systems, autopilots navigation and sighting equipment etc.[4] Today Saab Security & Defense Solutions Training & Simulation in Helsingborg is one of Saab's many branches, developing training solutions, mostly for military training. Still, some of their solutions are intended for civil use, one of which is their air traffic control training system, Saab ATC solution.

1.2.1 Saab ATC solution

The Saab ATC solution (Figure 1) is a training system in which air traffic control students may practice in a realistic, simulated environment. The student is seated at a desk inside the ATC trainer TWR that simulates the environment surrounding the air traffic control tower. Throughout an exercise the air traffic control tower student takes on the responsibilities of an air traffic controller. This means that the student has to react to situations like changing weather conditions and airplanes wanting to land. He or she must provide pilots with necessary information and direct them to avoid accidents and make efficient use of airspace.
During a training session, a number of IATS Clients are used. These are controlled by teachers or students that control airplanes and other scenario related entities throughout the scenario. It is with these teachers and students the air traffic control tower student communicates throughout the exercise as though they were the actual pilots.

The scenarios that students are to work through are first created by an air traffic control teacher in the scenario management. It is in the scenario management that new taxi routes are created and weather conditions can be defined. Once scenarios have been created in the manager a session can start for a student in the ATC Trainer TWR. These scenarios are saved for future use.

---

[3] Wise, John. Hopkin, David and Garland Daniel. *Handbook of Aviation Human Factors, Second Edition*. CRC Press 2009. http://theblackswaninvestmentclub.com/flight_manuals/human%20factors.pdf pg. 20-2 (Accessed 2013-05-21)
[4] SAAB AB. 2013, Saab Historic Milestones, http://www.saabgroup.com/en/About-Saab/Saab-History/Timeline/ (Accessed 2013-05-21)

It is the ATC Simulator (ATSsim) that holds the most intelligence in the Saab ATC training solution. ATSsim keeps track of the location of all aircrafts in a scenario and calculates their movements depending on speed, direction and other parameters.

WISE is the database that ties all these different parts together ensuring smooth communication between them.

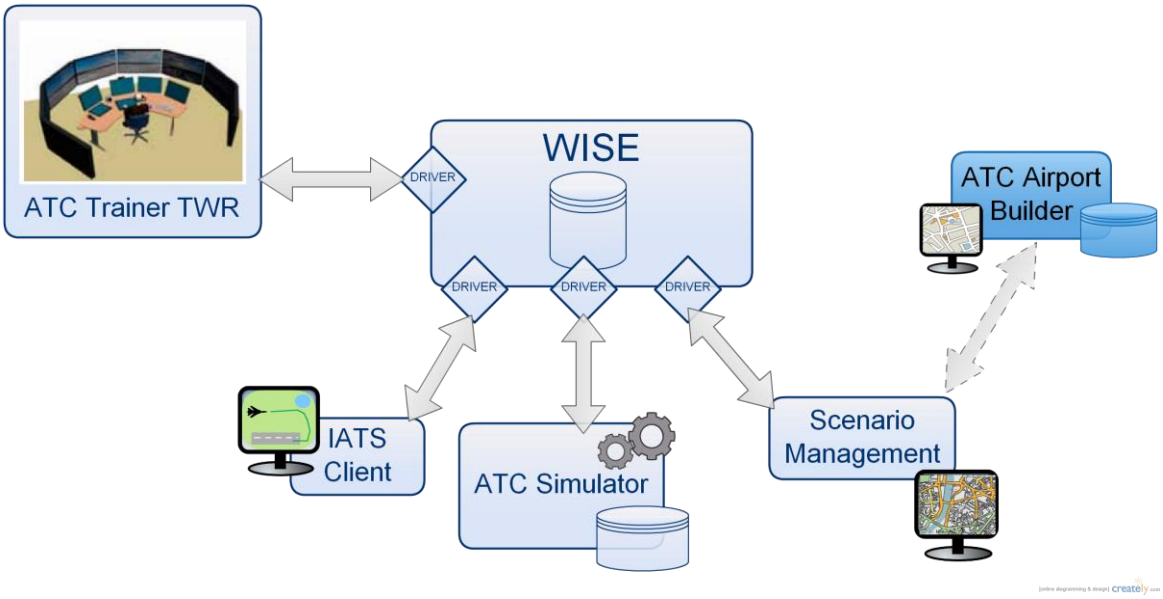


Figure 1: SAAB ATC solution

1.2.2 Windows Presentation Foundation

Today much of the Saab development in Helsingborg is done in .NET. For graphical interfaces Windows Presentation Foundation (WPF) is often used.[2] WPF is a Windows client based system for development of graphical .NET applications. Developing in WPF involves the creation of XAML (extensible application markup language) markup code, the backbone of the graphical elements and the creation of the underlying .NET code (in this case C#). One of the advantages of WPF is this division making it easier to divide work between a developer/designer specialized in the creation of user interfaces and the .NET developer. Visible user interface elements may be created in the XAML markup (the .xaml file) while all the run-time logic is created in separate files linked to markup objects.[5] XAML is in many ways similar to other markup languages such as HTML in combination with CSS allowing the developer/designer to use styles to change several graphical elements in one

[5] Garofalo, R. *Applied WPF 4 in context*. Berkley: APress, 2011. http://link.springer.com.ludwig.lub.lu.se/content/pdf/10.1007%2F978-1-4302-3471-5_1.pdf (Accessed 2013-05-21)

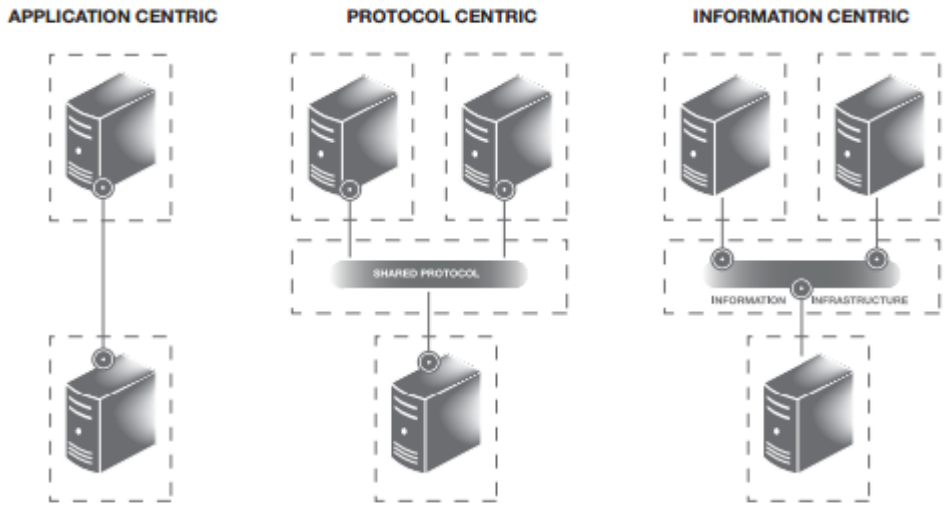
place. Separating appearance-specific markup and underlying code may also reduce maintenance time as changes in one may not interfere with the other.

Another advantage is the speed in which graphical interfaces can be created and modified with the help of XAML. WPF has default controls and containers with predefined graphical looks that can be used and/or modified with just a few lines of XAML code. This gives the developer/designer much freedom but also the tools to quickly create graphical interfaces.[6]

1.2.3 WISE

Widely Integrated Systems Environment (WISE) is an information-centric integration architecture combining the numerous applications within Saab (Figure 1). Before the development of WISE, the integration principle of SAAB training systems was application-centric, meaning that only two systems were involved and directly connected to one another. (Figure 2) This required both systems to be adapted to the same protocol. As the need to integrate more systems arose, so did the need for new integration architecture. The first step was to use one shared protocol through which data could be exchanged. This protocol-centric integration principle allowed for more systems to be integrated but at the cost of flexibility. Each application's end was connected to the integration system through one common protocol. Thus changes at one end of the system often resulted in necessary adoptions throughout all endpoints of the applications.[7]

[6] Microsoft, 2013, Introduction to WPF, http://msdn.microsoft.com/en-us/library/aa970268.aspx (Accessed 2013-05-21)
[7] SAAB AB, http://www.saabgroup.com/Global/Documents%20and%20Images/Land/Collaborative%20Training/pdf%20and%20thumbnails/Saab_WISE_Concept.pdf (Accessed 2012-05-21)



8

Figure 2: WISE evolution

In 2005 Tobias Almén at Saab Helsingborg started the process of taking the integration architecture of the system to yet another level. The result, released the same year became known as WISE. It is an information-centric integration architecture allowing for data exchange throughout the system regardless of the different application technologies. By moving the integration points away from the application into the information infrastructure, integration was no longer dependent on the specific technologies of the applications. Instead integration was dealt with through drivers at the WISE server allowing for enhanced system development.[2]

1.2.4 ATSsim

The ATSsim is the aircraft simulator of Saab's air traffic control training system written in C++. It contains no graphical interface as its main task is to simulate aircraft movement. Every tic (one second), the simulator performs all necessary calculations to find the current position of an aircraft depending on parameters, such as speed and direction in the WISE database. In reality, a tic may be divided into even smaller units where different calculations and checks take place. The ATSsim is aware of ground height, available landing points and positions of all aircraft. It has functionality to detect collision between aircrafts. That is as far as the ATSsim intelligence goes in terms of knowledge of spatial reference. The ATSsim can for example not detect a collision with a building, and allows takeoffs from oceans.[9]

[8] SAAB AB, [online image] http://www.saabgroup.com/Global/Documents%20and%20Images/Land/Collaborative%20Training/pdf%20and%20thumbnails/Saab_WISE_Concept.pdf (Accessed 2012-05-21)

[9] Lubera, Mathias; Software Engineer, Saab Training and Simulation. ATSsim, informal conversation, 2013-05-13.

1.2.5 3D engine

Saab's air traffic control training system 3d engine communicates with the system through WISE. It is the part of the system responsible for generating 3d environments. The 3d environment in the 3d engine must first be modelled manually by a 3d modeller (for example a SAAB employee or SAAB customer). This model within the 3d engine is separated from the 2d world rendered in the client. Hence, if the 2d world is changed, the 3d world within the 3d engine needs manual adjustments accordingly.[2]

1.3 Geographic Information Systems (GIS)

Geographic information has been stored for centuries as content on maps or observation records on paper. As geographic information was later also handled digitally on computers, providing possibilities for quicker processing and more precise computations, the interest in digital geographic information systems arose. In the mid-1960s GIS was created to achieve just this.[10]

The purpose of Geographic information systems is to describe the known positions of geographic objects on earth's surface. To define these positions in a uniform manner a coordinate system is used. Geographic coordinate systems specify a position by a set of numbers or letters, typically longitude and latitude and height. Latitude (φ) represents the north-south position and ranges from -90˚ to +90˚ where 0˚ is along the equator and longitude (λ) represents the west-east position and ranges from -180˚ to +180˚ where 0˚ is along the prime meridian (usually the plane through Greenwich but can be any arbitrary meridian).[11]

As a position is always relative to a reference, geodetic reference systems are used to specify this relationship. The ability to specify different reference systems allow for more precise measurements where needed, e.g. on local levels. It would however be ideal if the same system could be used world-wide.

The two-dimensional reference system RT90, which is based on the ellipsoid Bessel 1841 and the Gauss-Krüger conformal cylindrical projection, was the standard reference system in Sweden up until 2006, and is since being phased out.[12] The three-dimensional reference system SWEREF 99 replaced RT90 in

---

[10] Goodchild, M. F. *International Encyclopedia of Human Geography*, Santa Barbara, Elsevier, 2009, pp. 526-531, Available from scienceDirect (Accessed 2013-04-11)

[11] Eklundh, Lars. *Geografisk informationsbehandling: Metoder och Tillämpningar*, Västerås, formas, 2003 pp. 67-89.

[12] Lantmäteriet, 2013, RT 90-Lantmäteriet, http://www.lantmateriet.se/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/Referenssystem/Tvadimensionella-system/RT-90/ (Accessed 2013-05-08)

2007. It deviates approximately 0.5 metres from WGS84, the reference system used by the Global Positioning System, and can therefore be regarded as identical in many applications considering the fact that the uncertainty of WGS84 generally is about 10 metres.

To transfer geodetic coordinates to a flat surface, map projection techniques are used as described in[13] Unfortunately, it is theoretically impossible to create a perfect projection of an ellipsoid on a flat surface and therefore the result of all map projections is distorted in shape, area or distance. The coordinates are usually transformed using complex mathematical functions, but can generally be described as:

$$X = f_x(\varphi, \lambda)$$
$$Y = f_y(\varphi, \lambda)$$

Where φ represents geodetic latitude and λ geodetic longitude, and x, y are coordinates in the flat projection plane.

One of the iconic concepts of GIS, is the principle of layers which allows for data to be thematically organized in different layers. Another is the concept of vector schemes identifying each feature on a map. Each feature is categorized as either a point (represented by a pair of coordinates), lines/polylines (represented by a sequence of pints connected by straight-line segments) or polygons (represented by an area created by similar sequences connected in loops).[10]

1.3.1 ESRI shapefiles

The ESRI shapefile, which model was introduced in the late 1980's, stores spatial data features as points, polylines or polygons as well as their associated attributes.[10]

Today the ESRI shapefile is the most common shapefile standard. Each "shape file" is composed of several files out of which the three files .shp, .shx and .dbf, are mandatory.[14]

A more thorough analysis of the ESRI shapefile structure can be found in section 3.2.1.

---

[13] Lantmäteriet, 2013, Kartprojektionens grunder-Lantmäteriet, http://www.lantmateriet.se/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/Om-geodesi/Kartprojektioner/Kartprojektionens-grunder/ (Accessed 2013-05-08)

[14] ESRI, ESRI *Shapefile Technical Description* http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf (Accessed 2013-04-11)

1.3.2 Falken

Falken is a standard with specific instructions on how map data (for example the data in the ESRI .dbf shapefile) may be organized. The Falken1 document consists of specifications regarding map data, used by SAAB training system AB. The specifications have been gathered from several different sources out of which luftfartsverket AIP, HKV/Luftfart, Anläggningar(TBANK), FMV:Infosyst(GSD) are a few. Some of these specifications are relevant for this work.

The map data is divided into different layers where each layer has been assigned one shapefile. For each original layer only one kind of object is allowed. The name of the file is to reflect this object type. Sometimes these filenames are changed upon delivery to target system in order to reflect the facility, rather than object type. [15]

The different object types listed in Falken1 can be found in section 8.1.

1.3.3 GDAL/OGR

The Geospatial Data Abstraction Library (GDAL/OGR) released by the Open Source Geospatial Foundation is a cross platform C++ library for raster (GDAL) and vector (OGR) geospatial data formats. The library provides an abstract data model that is shared between the supported formats. Although the library is written in C++, the GDAL project also maintains generated C# bindings for use with the .NET languages. OGR supports over 20 vector formats.[16]

1.3.4 SharpMap

SharpMap is an open source C# library based on the .NET 4.0 framework that gives applications the possibility to access many types of GIS data and provides the ability to execute spatial queries on said data. The library also enables applications to render maps by providing developers with easy-to-use controls for this purpose.[17]

2 Problem description

The purpose of this thesis project is to find and test a solution in which data from 2d shapefiles can be displayed and modified in a way that it may later automatically create 2d airports in Saab's air traffic control training client. The

[15] Combitech, *Grundstruktur för kartdata i FALKEN1*, 2006

[16] OSGeo, 2013, GDAL/OGR Info Sheet | OSGeo.org, http://www.osgeo.org/gdal_ogr (Accessed 2013-05-08)

[17] Microsoft, 2013, SharpMap-Geospatial Application Framework for the CLR http://sharpmap.codeplex.com/ (Accessed 2013-05-08)

idea is that this data, modified in the thesis project prototype should be integrated with the ATSsim (described in section 1.2.4) for more intelligent aircraft simulations. Another vision is to also integrate this prototype with the 3d engine (described in section 1.2.5) in which by the help of further studies 3d environments may be generated through this data.

This involves a study of the structure of the ESRI shapefile to determine what data can be used in the ATSsim and 3d engines. In order to sort out what data is needed an understanding of the ATSsim, the 3d engine and airport management is necessary. Another aspect of this work is decisions in how to sort and store data gathered from the shapefiles.

Furthermore a graphical interface allowing users to view and modify shapefile data is to be implemented in a WPF (described in section 1.2.2) based .NET environment. This involves not only decisions on what libraries provide the best support for this implementation, but also the implementation itself. This graphical interface will from hereon be referred to as the ATC AirportBuilder Wizard. The ATC AirportBuilder Wizard and all modifications in communicating systems will be referred to as the ATC AirportBuilder.

Lastly, an analysis of how this graphical interface could best be integrated with the ATSsim will be made.

The following questions will be answered in this thesis:

- What data is contained within a shapefile? Which of this data can be used in the ATSsim and 3d engine?
- What data tables will be needed in the prototype to store airport related shapefile data?
- How can shapefile data be viewed and modified? How CPU-demanding is the method chosen for viewing shapefiles? What sort of modifications may be necessary?
- How can the extracted information from shapefile data be integrated with ATSsim?

2.1 Visions

By automating the process of airport creation in the 3d engine much time can be saved. This is a feature that could enhance the Saab products giving them a stronger stance among competitors. Also, the simulator ATSsim could benefit as shapefile data could be used for better scenario management. Saab wants to investigate how the ATSsim and 3d engine can benefit from the information stored in shapefiles and how it can be incorporated into their products.

Their vision is to use data in existing shapefiles to generate 3d airport environments. Another vision of theirs is to take advantage of shapefile data in the ATSsim to enhance simulator intelligence. Saab hopes for these visions to, in the future materialize into solutions for world-wide use.

2.2 Scope

2.2.1 Design of ATC AirportBuilder database

This thesis project will involve the design and implementation of the ATC AirportBuilder database.

This database shall fulfil the following criteria:

- Be able to store all information found in a shapefile that may be of use to the ATSsim and 3d engine.
- Store data that has been prepared to facilitate future integrations with the ATSsim and 3d engine.
- Be adaptable to meet a global market.

2.2.2 Design and implementation of the ATC AirportBuilder Wizard

This thesis project will involve the design and implementation of the ATC AirportBuilder Wizard.

This wizard shall fulfil the following criteria:

- Be able to graphically display the spatial content from shapefiles.
- Be able to manually modify area information (e.g. area type, description, height) through a user friendly interface.
- Be able to sort areas in a logical manner.
- Be able to prepare, store, retrieve and delete information in the ATC AirportBuilder database.

2.2.3 Performance test of shapefile display technology

This thesis project will involve a performance test in which the ATC AirportBuilder Wizard's shapefile display technology is tested.

This test shall fulfil the following criteria:

- Be able to determine how much CPU is needed by the technology to display 400 moving points.

2.3 Limitations

The ATC AirportBuilder shall serve as a prototype and not as a finished product. With future developments that lay outside the scope of this thesis, a commercial product may be finalized.

This thesis project does for example not involve the integration between ATC AirportBuilder Wizard, ATSsim and 3d engine. Nor does it provide specific

specifications for how it should be integrated. Instead it gives some broader guidelines of how the integration task could be approached.

In the future it would be desirable to be able to modify the spatial data of an area within the ATC AirportBuilder Wizard. This feature lay outside the scope of this thesis project as well.

The performance test conducted in this thesis project is not to provide a comparison between different similar technologies. However it does provide some idea of how well or even if the technology would suffice when used in more demanding circumstances.

3 Method

3.1 Project model

Since the scope of this thesis project involves many, to the authors, new concepts, the thesis project is divided into smaller, more manageable tasks. These tasks allow for flexible decision making where choices regarding the next upcoming part can be made based on results of previous ones. Hence, many pitfalls may be avoided.

Figure 3: Project Model chart

In order not to spend time on one task for too long a project model is created. (Figure 3) This is also a way to gain an overview of the thesis project. The thesis project is carried through with a quick iteration through each task leaving flaws and unfinished issues. One final polishing cycle (The circle of large blue arrows in Figure 3), iterating through the entire thesis project, leaves room for corrections and smaller changes. This workflow allows for some errors and mistakes that can be corrected during the final iteration. Being able to use methods of "trial and error" is crucial when breaking new ground and trying techniques unknown to the authors.

Another important part of the model chosen is the daily use of notebooks in which thoughts, problems and decisions are noted in an informal manner. These notes are composed by both authors individually. They are saved for future reference, and also to create a platform for discussion and reflection. Through daily meetings based on these notes, short and long term tasks are prioritised and revised.

3.2 Project structure

The structure in which this project has been performed is divided into five tasks;

1. The analysis of the shapefile.
2. The database model and data analysis.
3. The implementation of the ATC AirportBuilder Wizard.
4. The analysis of the ATC AirportBuilder Wizard technology performance.
5. The analysis of the integration between the ATC AirportBuilder Wizard, the ATSsim and 3dengine.

Each task requires information gathering which is performed before the start of each new phase. As the need arises for more information during a phase, occasional breaks in the process can be made to fill these gaps.

The tasks are ordered in chronological order of when they are to be performed (with the exception of task 3 and task 4 that are performed in parallel to one another). This is done because of the dependence relationship between them where each task is dependent on the results of the previous one (Again, with the exception of task 3 and 4 that have a mutual dependency relationship). It is these dependences that lay the boundaries for the order of prioritisations of the project. Since most tasks are dependent on previous tasks, a prioritisation scheme congruent to chronological order becomes an obvious choice.

3.2.1 Shapefile Analysis

The first task is to find a way to access all data in a shapefile. The analysis also involves some research in terms of how the spatial data is stored within the file to ensure correctness of future data interpretation. Furthermore, the remaining shapefile data is to be investigated to bring light to useful data.

3.2.2 Database model and data analysis

The database model is dependent on both the data in the shapefiles and the information gathered regarding airports and the ATSsim and 3d engine. When the information needed is gathered and after a discussion with the advisor at Saab a database is created in Microsoft SQL server. This database contains tables for those data from the shapefiles needed in the ATSsim and 3d engine.

3.2.3 Implementation of ATC AirportBuilder Wizard

This part of the thesis project is the most time consuming as it involves the implementation of a prototype (ATC AirportBuilder Wizard). It also answers the third question that links most of the other answers together. The ATC AirportBuilder Wizard provides an interface to fill in possible gaps where shapefile data is inadequate to meet the needs of future components.

This task involves many steps starting with viewing the graphical information contained in a shapefile inside a .NET WPF application.

It also involves the making of a graphical .NET interface in which the information in shapefiles can be modified and stored to the database. This means decision making in terms of what sort of shapefile modifications may be necessary and how this data is best stored to the database. Decisions regarding what libraries to use for these modifications also have to be finalised at this stage.

On top of this Saab also wants an estimation of how CPU-demanding the method of choice is for viewing shapefiles. This demands the setup of a test-case where performance of the graphical interface is tested at different workloads.

3.2.4 Performance

The performance of the chosen method for viewing shapefiles is important to take into account, firstly because it needs to be fast enough for use inside the ATC AirportBuilder Wizard, but it is also important from a broader perspective. It is important to Saab to see if the chosen method is performing sufficiently well to be reusable in future applications.
This demands the setup of a test-case where performance of the method is tested at different workloads.

3.2.5 Integration

One vision of this thesis project is to enable future automatic 3d mapping from shapefile data to the air traffic control training systems. This requires the ATC AirportBuilder to be integrated with not only the simulator but also the 3d engine. Such implementation of the prototype lies outside the scope of this thesis project. However, a smaller analysis of how this implementation could be done in the future is done during the integration part of the thesis project.

3.2.6 Information gathering

At the start of the thesis project a class is taken on the basics of air control management. It provides some of the necessary understanding of how air traffic control management works, hence what parts are important in a training simulation program. The class also provides an understanding of the domain specific terms linked to air traffic control management.

The material listed in the footnotes is read throughout the project as the need arises. At times the Saab advisor and other employees at Saab are consulted for advice and further insights. This information provides the backbone for decision making in the thesis project.

Also a lecture is given by Mathias Lubera, the advisor at Saab, about WISE and how it is connected to the reset of Saab's ATC solution. Notes from this occasion provide valuable help in understanding the basics of the system communications.

Throughout the entire thesis project, Lubera and other Saab employees have provided valuable input through discussions and by answering questions.

3.3 Source criticism

In this thesis project the aim has been to use many reliable and independent sources and to when possible check primary sources. Also, when possible, information from different independent sources has been compared to ensure reliability. Unfortunately this has not always been achievable. This thesis contains much information from a variety of fields ranging from air traffic control to the specifications of the ESRI shapefile. Some of these fields being rather narrow, finding several sources provided by independent objective sources proved at times not possible. It is worth noting that some of the documentation that has been used was provided by larger commercial corporations such as Microsoft and Saab. Organisations, corporations and other sometimes bias groups/individuals may according to Kristina Alexanderson produce facts where their own interests are incorporated, thereby weakening the reliability of the facts.[18] This is something that has been kept in mind throughout each research phase of the thesis project. While awareness may help to avoid unreliable facts from slipping through, it is no guarantee.

Another observation that may bring some concern is that the vast majority of sources in this thesis are gathered from internet sources.

4 Analysis

4.1 Shapefile Analysis

Since the common shapefile standard is created by ESRI the search for ways to examine these files started with ESRIs own products. Free trial versions of the software ArcMap and ArcCatalog were used to open and view shapefiles from one of Saab's fabricated, virtual airports; VIRUM.

Some research was done concurrently for a better understanding of what was seen. Some of these findings are discussed in this section.

[18] Kristina Alexanderson. 2012. Källkritik på Internet, https://iis.se/docs/Kallkritik-pa-Internet.pdf (Accessed 2013-05-21)

Each shapefile contain the following files;

.shp is the “main” file containing all the geographical information.
.shx is the “index”-file containing indexing allowing for quick access of objects.
.dbf is the “attribute”-file containing miscellaneous data as attributes. One example of an attribute is: “ObjectType: Water.” [14]

In order to organize the different spatial information within the .shp file in a logical manner that is always interpreted the same way standardisation is essential. Each .shp file may only contain one shape type (specified in the shape file record) out of (null shape, point, arc, polygon, multipoint). The geometric data for the shape is followed directly after the shape type specification. This geometric data is stored in the shape file record according to shape type;

A null shape, represented by #0, has no geometric data.

A point shape, represented by #1, consists of a pair of coordinates X and Y.

A multipoint shape, represented by #2, consists of a set of points. The data is structured as follows: First the bounding box (4 double values giving Xmin, Ymin, Xmax and Ymax of all points), then an integer giving the total number of points and lastly the points in the shape.[19]

An arc shape, represented by #3, may consist of multiple PolyLines (ordered set of vertices). These PolyLines may or may not be connected to one another. The data is structured as follows: First the bounding box, then an integer giving the total number of PolyLines, then an integer giving the total number of Points, then an array of length NumParts. The array stores the index of the first point for each PolyLine in the points array. Lastly, there is an array of length NumPoints.

A polygon shape, represented by #4 consists of a number of closed, non-self-intersecting loops defined as rings.[19] Since a polygon may contain multiple outer rings, a way of defining what is part of the polygon area, and what is a “hole” in the area becomes essential. ESRI has defined this as:

“The order of vertices or orientation for a
ring indicates which side of the ring is interior of the polygon. The neighbourhood to the

[19] ESRI, ESRI *Shapefile Technical Description*, pp. 4-6.
http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf (Accessed 2013-04-11)

right of an observer walking along the ring in vertex order is the neighbourhood inside the polygon. Vertices of rings defining holes in polygons are in a counterclockwise direction. Vertices for a single, ringed polygon are, therefore, always in clockwise order."

The structure of the polygon file is identical to the one of the arc shape.

Apart from spatial data, the ESRI shapefile also allows for storage of attributes, stored in the .dbf file.[14]

Spatial information in the files could be viewed using ArcMap, where each .shp file could be displayed as a separate layer. By opening the layers attribute table the data contained within the .dbf files could also be viewed. Most of the VIRUM .dbf files contained the following table: (Figure 4)

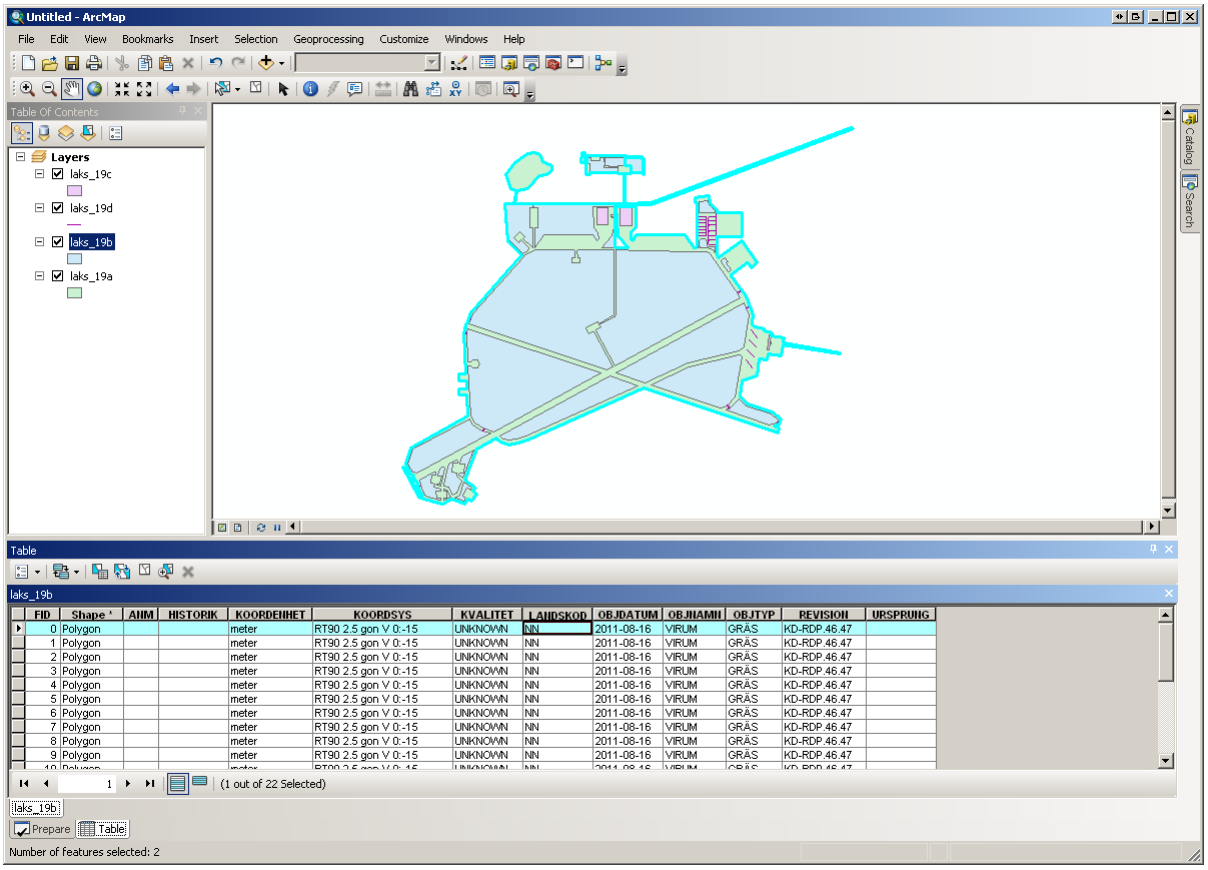


Figure 4: VIRUM files viewed through ArcMap

This information given by ArcMap was also compared to the information given from a third party program, "DBF Viewer plus". The information from both programs was found to agree: (Figure 5)

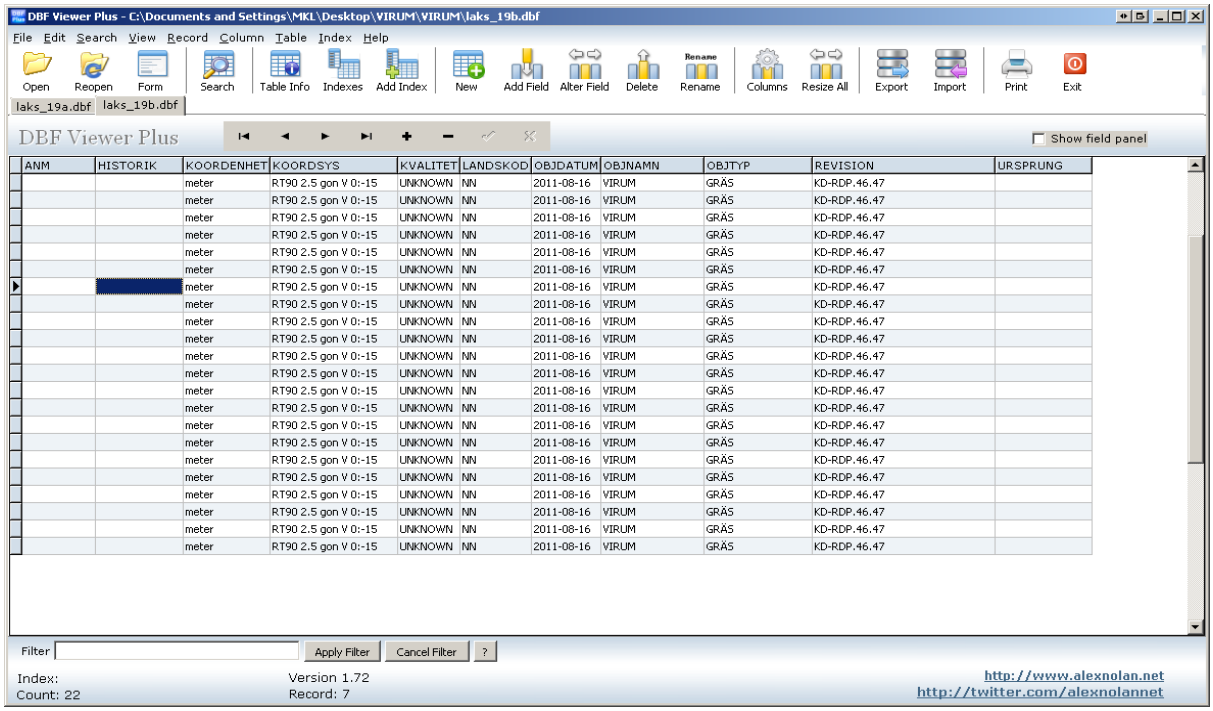


Figure 5 : VIRUM .dbf files viewed through DBF Viewer plus

In order to later manipulate shapefile data, the raw spatial data was also needed. This is information that claimed impossible to gather through these programs without spending further time learning about their specific functionalities. In accordance with the project model decisions were made to progress without these findings. The fact that the ESRI shapefiles were so well documented, providing this information, further supported this decision.

One thing found in the ESRI shapefile documentation that proved important is the way “holes” were represented in the data. (p. 21 Figure 7) This information was to be important later when designing the database.

Another important finding is the possibilities of varying .dbf file content. In contrast to the .shp file content the content of the .dbf file is undefined, allowing different kinds of fields and data.[14]

In order to, at a future point, make 3d models based on the information from the 2d-shapefiles, height data is desirable. As expected height information is not provided in the sample files. However, most of the VIRUM shapefiles contained a field called “OBJTYP” (object type) in the corresponding .dbf file. Different OBJTYPs were for example “GRÄS” (grass), “BYGGNAD” (building) and “BANA” (runway). While this information does not directly provide height specifications, some general conclusions can be drawn. Considering the main use of the 3d engine which primarily serves air control training purposes, these general conclusions may give sufficient accuracy. If the ATC AirportBuilder software could utilize OBJTYP data, translating this

information into height and rendering information according to a given scheme, much tedious work could be saved by this application.

One problem with this approach is that not all .dbf files contain “OBJTYP” data. This is likely to happen in those cases where the Falken standard (described in section 1.3.2) is not followed. If for some reason this data is missing, the application needs an alternative way of dealing with height and rendering aspects. This omission of the “OBJTYP” field and data is possible due to the generous .dbf content rules. Since the fields in the .dbf file can contain infinite combinations of different fields with data, or even none at all, finding a solution becomes important. Although, the Falken standard might provide an answer, this is hardly lasting. Even within the VIRUM files (that supposedly follow the Falken standard) deviations were found. Also, Falken is a Swedish standard, with Saab being a highly internationalized company basing an application on a Swedish standard, using definitions in Swedish, quickly becomes limiting. As one of the future goals of this thesis project is to provide a less labour-demanding way off adding new airports to air traffic control training systems world-wide, allowing for many different standards seem a better choice. A decision was made to take these “issues” into account during the implementation phase, where they can be tended to.

Another possible problem that could be seen when analysing the VIRUM shapefiles was that the spatial information was minimized. Some “holes” in the areas were omitted from the spatial data in certain files. This is for example clear in the runway (shown below to the left, Figure 6) which is represented by a large field rather than pathway-shaped area when displayed solitarily. Only when analysed in a context together with intersecting spatial data (shown below to the right, Figure 6) did the exact paths of the runway become visible. This is another issue that will need to be addressed in future versions of the ATC AirportBuilder.

Figure 6: Graphical interpretations of VIRUM shapefile data. From left to right, road area, grass area, road and grass area together.

4.2 Database model and data analysis

The first part of this task involved some research. An example database from ATSsim, the data model of the simulator was studied with hopes of providing guidelines in how to best arrange data. Since the ATC AirportBuilder Wizard database is later to be used with ATSsim, congruency between the two is desired. However, this study proved futile as most data in the data model of the simulator turned out to be irrelevant to this study. The data model of the simulator database contains mainly data representing moving entities such as airplanes, whereas the ATC AirportBuilder Wizard database primarily contains data representing objects from the environment.

Instead of using the data model of the simulator for guidance focus was then turned to the air traffic control lecture notes and discussions with the advisor at Saab. A decision was made to divide the data to represent in the database in two general groups, "airport-related" and "area-related" data.

4.2.1 Airport-related data

The airport table serves as a hub tying all areas related to a certain airport together. It can also store important airport-related data. At first, storing the airports specific airport code as the ID and private key may seem like a good idea. But as the air traffic control class had shown, some airports may lack airport code and duplicates may possibly occur. Therefore, this idea is discarded.

In the air traffic control class, country codes were drawn to attention as important. Therefore this country code information may be useful in the airport builder database. However, this information proved redundant since country code information can be gathered from the airport code.

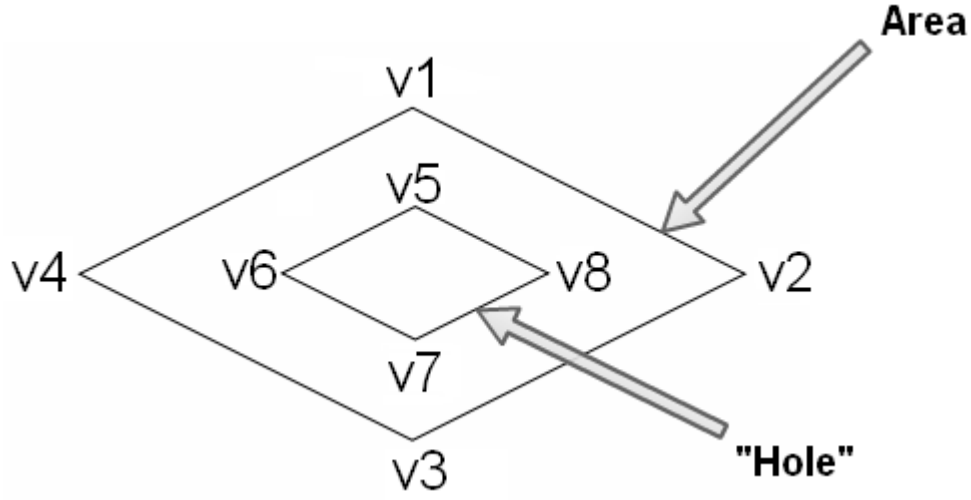
Since there are several different standards for coordinate systems, storing the coordinate system in the airport table may seem like a good idea. All areas contained in an airport need be of the same system or they will mismatch in a graphical display. However, upon researching the ATSsim we found that the ATSsim only stores coordinates in the WGS84 standard. To ensure congruency between the different platforms a decision was made to discard having a coordinate system field in the airport builder database. Instead all spatial data not already in the WGS84 format will have to be converted into the former before entering the airport builder database. Another reason for sticking to the WGS84 format is that it is internationally recognized as opposed to the RT90 format (a Swedish system) used in for example all the VIRUM shapefiles.

Apart from the help from the air traffic control class, discussions of database design with the advisor proved highly valuable. One point brought to attention during such a session was the need for different airport resources that may be tied to and shared between several airports. An example of such resource may be a fire truck.

4.2.2 Area-related data

The data most important to the ATC AirportBuilder is the Area-related data since this data includes all spatial information.

The first issue discussed was how to represent "holes" in areas. A Boolean field flagging for a "negative" (hole) area segment was first suggested.



[20]

Figure 7: An area with a hole

But since the shapefile analysis had shown that this information already exists in the .shp coordinate data this Boolean flag would be redundant (given that the .shp coordinate data is stored in a similar manner in the ATC AirportBuilder database). The points in a ring shape in the .shp file may be organized in either clockwise or counter-clockwise order. It is this order that determines whether the ring shape represents an area itself or a "hole". At first thought, it may seem cumbersome to go through all points in a ring shape to see if they are organized in clockwise order or not. On the contrary, this is a simple task that can be handled through well documented algorithms.[21] There is therefore little reason to use a Boolean flag. In this thesis project the decision was made to rely solemnly on information about order of points and the use of the algorithms mentioned.

[20] ORACLE, [online image] http://docs.oracle.com/html/A88805_01/sdo_objl.htm (Accessed 2012-05-24)
[21] Microsoft, 2013, SharpMap – Geospatial Application Framework for the CLR – Source Code, http://sharpmap.codeplex.com/SourceControl/latest#2058263 (Accessed 2013-05-15)

Other data that was considered for the database was max values for a shapes x- and y-coordinates. These values could be used to trace midpoints of an area for accurate placement in a map context. As efficient algorithms to find these values from .shp coordinate data, extra fields were found redundant.

Since these efficient algorithms exist, the simplest way of saving the .shp coordinate data is to convert the .shp file data to "Well-known text format"(WKT). WKT is a text format representing vector geometry and for making transformations between different spatial reference systems. This WKT text data could then be saved to the database as one long string, keeping its current structure intact. One problem with this method remains. The use of this method allows for many different shapetypes, some of which may not be supported in future ATC AirportBuilder functionality. More control is granted to the ATC AirportBuilder software if only a few necessary, predefined shapetypes were allowed. This can be achieved with the use of an extra database table, "Shapetype," with a relationship to the "Shape" table. This requires each shape to be saved as a separate entity within the ATC AirportBuilder database. Hence the coordinate string originating from the .shp file must be split into shapes.

This method resolves the issue of unknown shapetypes. It does also create redundancy in the ATC AirportBuilder database unless the data from the .shp file is further modified before entering the database. The piece of text introducing each shape may now be omitted.

A different way of saving .shp file spatial data to a database is to use table-valued parameters. This SQL server 2008 feature allows for several rows of data to a stored procedure of function, excluding the need for extra tables. The evaluation of this alternative method lies outside the scope of this thesis.

As discussed in 4.1, the .dbf may include numerous tables of useful information. Even if this data sometimes is missing or not following the Falken standard (discussed in 1.3.2) and therefore not used directly by the application (all depending on choices made in the implementation phase) it may still be worth saving to the database.

4.3 Implementation

The first step in implementing the ATC AirportBuilder Wizard was examining if useful solutions existed at Saab that could be easily modified. The main obstacles to be tackled were reading a shapefile and extracting its spatial and .dbf data, preparing this data for storage in the database, as well as transforming the data from mere coordinates into something that could be drawn on the screen. It would also be of interest to be able to perform

calculations on the layers as possible difficulties in the way layers relate to each other had occurred when examining the spatial data, (described in section 4.1) It could for example be useful to be able to erase one layer from another.

4.3.1 Examining libraries

As Saab's existing simulation environments already possess the functionality to display overviews of airports, ideas of using this functionality in the prototype were exchanged with the advisor at Saab but proved not to be a useful way to progress. The main reasons why this would be a bad solution was that the simulation environments were both complex, and written in the language C++, which would result in time consuming work not only to locate the useful code in the system, but also to modify it so that it could be used in the environment of the ATC AirportBuilder.

The discussions with the advisor led us to instead investigate if any useful libraries were available online which could be used to satisfy the requirements the prototype was to fulfil. After thorough examination of several libraries, it was primarily two that seemed fitting for the purposes of the prototype; GDAL/OGR (described in section 1.3.3) and SharpMap (described in section 1.3.4). GDAL/OGR has been around for many years and seemed like a robust foundation for our prototype to build upon. SharpMap is very easy to use, and could be used almost out of the box. Another reason why these libraries were chosen was because they are both released under the terms of generous license agreements. GDAL/OGR is distributed under the terms of X11/MIT License and SharpMap under the GNU Lesser General Public License. GDAL/OGR would be used to extract data in shapefiles, perform calculations on this data, and prepare the data for the database.

SharpMap, which is built on GDAL/OGR and has the same functionality, would be used to render the data, while other functionalities would be provided by GDAL/OGR. The reason for this separation of functionality was to make it easy for future developments and modifications. As it was unknown at the time of this thesis project if Saab wanted to rely on the use of SharpMap when creating the final application based on the prototype, it was decided to try to minimize the use of SharpMap to only render the data on the screen. The result of this separation is that Saab with ease can discard SharpMap without having to rewrite all other code in the prototype.

4.3.2 Graphical user interface

After establishing what libraries to be used, a graphical interface was designed. The general idea was to end up with an application with a wizard-like feature that would produce a new airport along with associated areas. The necessary steps were defining the airport information, selecting shapefiles with information on the airport areas, displaying the data, and finally saving it to the database.

The most important bit of the interface was the window where shapefile data were to be displayed and modified. The goal of this window was to provide a graphical representation of the shapes, an overview over the current shapes in the scene, and a way to view and change the necessary properties of the shapes that had been opened, i.e. the .dbf data. As no presumptions could be made of what attributes the .dbf files contained, since attribute consistency cannot be guaranteed (see section 4.1), the best solution was therefore to let the user enter this information into the application. By displaying the .dbf data in the context of the map, the user would most likely be able to fill in the necessary information, such as "Area type".

In future versions of the ATC AirportBuilder, usability may be enhanced if .dbf attribute data is standardised. More on this in section 4.1.

4.3.3 Library integration

SharpMap is a library written for use with Windows Forms. Windows Forms is included in the .NET framework and enables developers to easily create applications for interacting with the user by providing access to familiar Microsoft Windows user interface controls. As the prototype was to be created using WPF, this was a problem. Even though branches of SharpMap with support for WPF was available, their license agreements made them impossible to use in the prototype, and the reliability of these branches was questionable. Luckily, WPF provides a Windows Forms host control, used to host existing Forms controls inside of a WPF application, which was used to host SharpMaps MapBox control.

The SharpMap MapBox control is a container for the Map control with many built-in functions for user-interaction, such as panning and zooming. The Map control holds a collection of Layers where each layer holds its own collection of Geometry objects which in turn can be created from the WKT (See section 4.2.2) of a shape by a simple method call. The Map will render all the layers added to its Layer collection and the MapBox enables user interaction with the map.

In this thesis it was decided to describe how OGR was used is by demonstrating a code example. To create an Area object along with its Shape objects from a shapefile using OGR, the following code was used:

```
public static List<Layer> GetLayersFromShapefiles(string[] files)
{
    Ogr.RegisterAll(); // Register all drivers for opening files

    List<Layer> layerList = new List<Layer>(); // Create list of layers
    foreach (string file in files)
    {
        try
        {
            DataSource ds = Ogr.Open(file, 0); // Open "file" no update required (0)
            layerList.Add(ds.GetLayerByIndex(0)); // If only 1 file is opened, there will only be 1 layer
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }

    }
    return layerList; //return list
}

public static Area CreateAreaFromLayer(Layer layer, Airport airport)
{
    return CreateAreaFromLayer(layer, layer.GetSpatialRef(), airport);
}

public static Area CreateAreaFromLayer(Layer layer, OSGeo.OSR.SpatialReference srs, Airport airport)
{
    OSGeo.OSR.SpatialReference srs_wgs84 = new OSGeo.OSR.SpatialReference(""); // Create empty reference
    srs_wgs84.SetWellKnownGeogCS("WGS84"); // Set WGS84 Spatial reference
    Area a = new Area(); // Create new area

    // Coordinate transformation
    OSGeo.OSR.CoordinateTransformation transform = new OSGeo.OSR.CoordinateTransformation(srs, srs_wgs84);

    // Create Shapes

    for (int i = 0; i < layer.GetFeatureCount(0); ++i) // For each shape
    {
        Shape s = new Shape(); // Create new shape
        Geometry geo = layer.GetFeature(i).GetGeometryRef(); // get geometry reference
        geo.Transform(transform);

        string wkt; // POLYGON ((..),(..))
        geo.ExportToWkt(out wkt); // Export to wkt
        string type = wkt.Substring(0, wkt.IndexOf(' ')); // Extract type from wkt;
        string coords = wkt.Replace(type + " ", ""); // Remove type from wkt. Left is only the coordinates

        s.shapeType = type;
        s.coordList = coords;
        s.Area = a; //Foreign key area
    }

    FeatureDefn def = layer.GetLayerDefn(); // Get layer definition
    a.name = def.GetName();
    a.height = 0;
    a.Airport = airport; // Foreign key airport

    return a; // Return area
}
```

Once an Area had been created in the prototype, the following code was used to create a SharpMap Layer that could be rendered.

```
SharpMap.Layers.VectorLayer layer = new SharpMap.Layers.VectorLayer(a.name); // Create
a new layer with area name
SharpMap.Data.Providers.GeometryProvider provider = null; // Provider for layer

foreach (var s in shapes)
{
    var geo =
    SharpMap.Converters.WellKnownText.GeometryFromWKT.Parse(String.Format("{0} {1}",
    s.shapeType, s.coordList)); // Create geometry from shape wkt

    if (provider == null)
        provider = new SharpMap.Data.Providers.GeometryProvider(geo); // Init provider
    else                                                               on first loop
        provider.Geometries.Add(geo); // Add geometries to provider
}
layer.DataSource = provider; // Set datasource to provider
a.Layer = layer;
```

4.3.4 Database handler and data model

When the reading and viewing functionalities of shapefiles had been implemented to the prototype, the fundamental bit of structuring shapefile data and saving it to the database was left to be implemented. In the very first version of the data model, plans were made to create class representations for each of the tables in the database. For example one Airport class and one Area class etc. where each instance of these classes represent a database table row. A database handler class was also planned to serve as the main interface between application and database. This structure, using a database handler, would make it easy to map objects in the prototype to the database. It would also simplify the recurring communication between the ATC AirportBuilder Wizard database and the wizard's general code, resulting in increased control and maintainability.

After reviewing the options available it was decided to dismiss the ideas of a database handler and use LINQ (Language-Integrated Query) instead. Using LINQ had many benefits; the most obvious, automatic generation of all necessary database classes, including classes representing tables in the database.[22] Thanks to C#'s "partial" keyword which is used to define these generated classes, the definition can be split into several source files. This makes it easier to manage these classes if property or method modifications are required, as these modifications can be separated from the generated code.[23]

[22] Microsoft, 2013. LINQ (Language-Integrated Query), http://msdn.microsoft.com/en-us/library/vstudio/bb397926.aspx (Accessed 2013-05-14)
[23] Microsoft, 2013. Partial Class Definitions (C#), http://msdn.microsoft.com/en-us/library/wa80x488(v=VS.80).aspx (Accessed 2013-05-14)

Another benefit was that the results of queries to the database were automatically translated into instances of the earlier mentioned classes, instead of returning an abstract data type representing rows and columns, such as Dataset. New instances of these classes could also be created and modified, and with a call to a method it could be saved to the database.[22]

4.3.5 Spatial references and coordinate transformation

As the spatial data was supposed to be stored in the WGS84 format in the database, it was essential that the prototype could transform coordinates described in different formats into the WGS84 format. To do this, spatial reference systems were needed. A spatial reference system specifies the coordinate unit, the ellipsoid, and the projection used to describe positions. This information is necessary when transforming coordinates from one system to another. Shapefiles can contain information regarding what spatial reference system their coordinates are described in[24]; however, this information is not always specified. In the case of the VIRUM shapefiles, the spatial reference system used was instead described by a string attribute in the .dbf file with the name of the projection. As mentioned earlier, the information in the .dbf file could not always be counted on, as attribute consistency could vary drastically. Therefore, one step in the wizard was dedicated to defining what spatial reference system was used for the opened shapefiles. Very much like what was done with the area types (described in 4.3.2), the attribute data was presented on the screen to help the user select the correct spatial reference system used for the shapefiles. When the spatial reference system had been set for the files, OGR was used to convert the coordinates into WGS84.

4.4 Performance analysis

A performance test of SharpMap was conducted to see how suitable SharpMap was to be used in future developments of Saab's other simulators. More specifically it was the SharpMap Map controls ability to render a high number of points on the screen that was to be tested.

The performance test case was primarily based on a description given by the Saab employee Jakob Blomberg. The goal was to display 400 points on a map and to move each point by a small degree each hertz to simulate the movement of airplanes. The CPU usage would during this test be monitored to see how CPU demanding this process was.[25]

---

[24] ESRI, Fundamentals of a shapefile's coordinate system, http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//00560000000q000000.htm (Accessed 2013-05-25)

[25] Blomberg, Jakob; Software Engineer, Saab Training and Simulation. Performance analysis, conversation, 2013-05-06.

In section 4.3.2, it was explained that SharpMaps Map control holds one collection of layers. This is not entirely true, and in this section, a deeper understanding of how SharpMap works is necessary. Therefore a more in-depth explanation of how the SharpMap layer collections work is provided here.

SharpMap holds three layer collections; the BackgroundLayer collection, the Layer collection, and the VariableLayers collection. The reason to have three collections of layers is to be able to differentiate between layers that are static and layers that change frequently. This is an important performance aspect that can be explained by studying this simple example:

Let us say that we would like to view a map with airplanes, represented by points, flying over Sweden. In this setup, a number of layers would be used, one layer with the outline of Sweden to be used as background, and to keep it simple, we put each airplane in its own layer. Now, as the airplanes fly around, their layers will be changed (as the points that represent them change position), and thus need to be re-rendered. This could be done in different ways. We could iterate through all layers and render them, which would work, but would be inefficient. That is because the layer with the outline of Sweden has not changed, and does not need to be rendered again.

This is where the three collections come into play. If the outline layer is put into the BackgroundLayer collection and all airplane layers are put into the VariableLayers collection, an image of the static layers (the outline layer) can be cached, while the variable layers (airplane layers) can be rendered on top of the cached image. The static layers will only need to be rendered once the area displayed in the Map control changes.

This is why the ability to differentiate between static layers and variable layers can lead to a decrease in processing power consumption.

The graphical user interface of the test case consists of a window which contains two buttons for starting and stopping the simulation and a Windows Forms Host control to host a SharpMap MapBox control. One layer with a polygon with a black outline is added to the Layers collection of the Map, to give a reference of how the points move around. The VariableLayers collection contains one layer which data source is set to a geometry provider. In this way, Points can easily be added to the geometry provider object, and the map will automatically render these points when the rendering method is called.

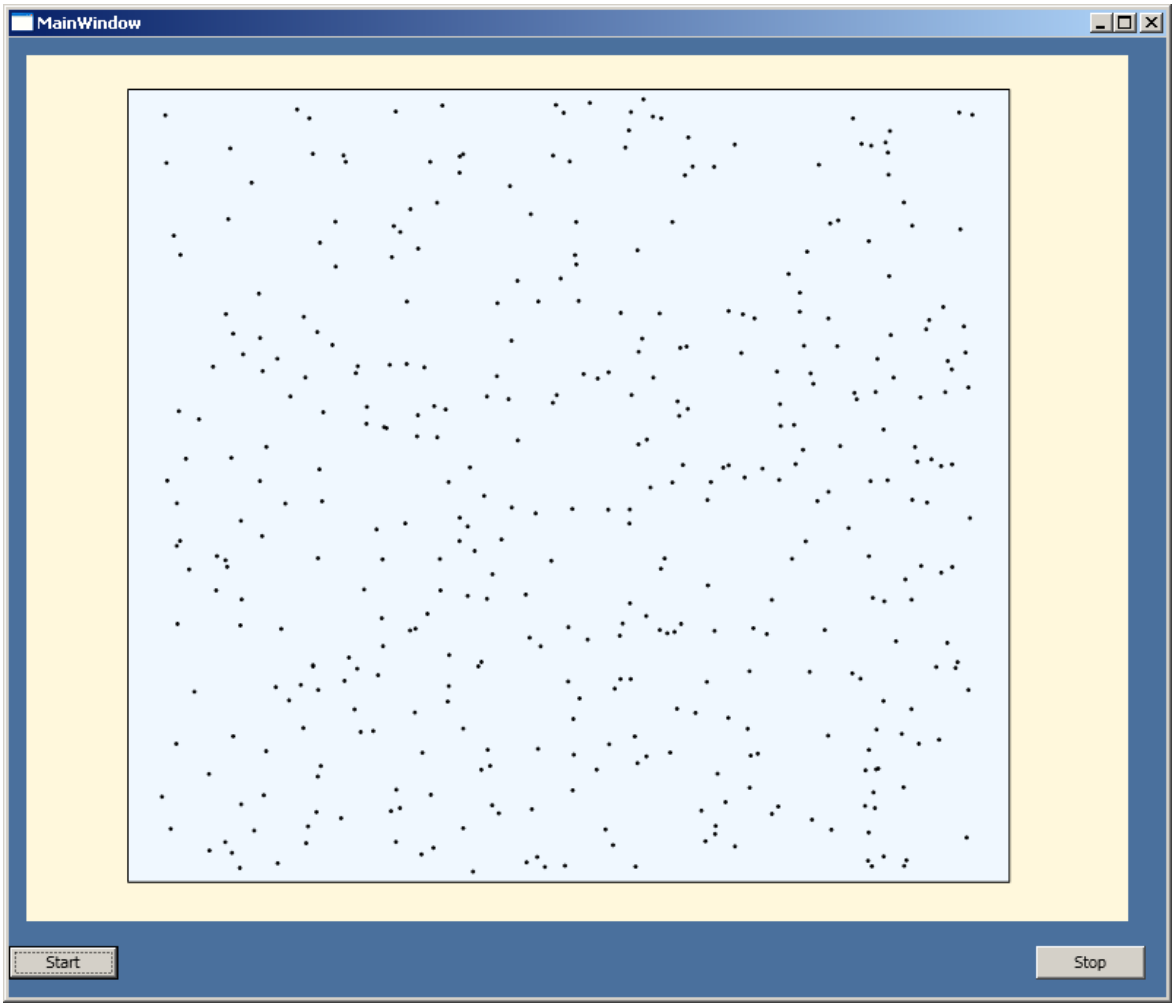


Figure 8: The graphical interface for the performance test. Points can be seen as black dots on a light blue background.

A Timer object is used to call a method every second. The method iterates through the points, adding random values to their coordinates, and calls the rendering method.

It is worth noting that in a real Saab simulation the airplane positions are calculated at a higher interval than one second. However, the positioning calculations would be performed at a server and not by the computer where airplane positions are displayed. Therefore, a higher interval would add unrelated computations and possibly distort the result.

In this test case, the TouchTimer method is called on the VariableLayers collection once every second to render the new positions of the points (Seen in Figure 8). The test was also conducted using the Refresh method of the MapBox control. There was no significant difference between the two, which may be due to the background layer being simple to render. With a higher number or more complex static layers, the difference between the two methods is expected to be bigger. Exactly what is the cause of this difference is not precisely determined in this thesis project, as this would involve

examining much of SharpMaps internal code, which lies out of the scope of this thesis.

4.5 Integration analysis

The first step in the process of integrating the prototype with Saab's ATC solution is the integration between the prototype and ATSsim. Once this is achieved, adaptions can be made in order for the 3dengine and the user client to support ATC AirportBuilder features. A reason for starting in this order is that ATSsim is the main part of the Saab's ATC solution, containing important simulation intelligence. While the 3d engine on the other hand rely on communication with WISE to render appropriate views.

4.5.1 ATSsim integration

In order to maximize the benefits from the shapefile data the integration between prototype and ATSsim should be given some thought. Currently the spatial data available to the simulator is restricted, limiting the simulator in making intelligent choices. This means that in order to for example have an aircraft run along a taxi-route, first the route needs to have been manually plotted, as ATSsim is oblivious to roads, buildings and other area types. If there are several different taxi-routes that only slightly differ from one another, each individual route still needs complete manual plotting. If instead the simulator could identify different areas and their types, such as the area of a road, well-designed algorithms could trace down possible taxi-routes along this road. This allows for countless possibilities in terms of future simulator capabilities. This would also take a heavy load off the administrators of the system, and Saab employees creating airport environments.

Collision detection between aircrafts and, buildings and other obstacles is another idea that can be implemented. Aircrafts could then make decisions to park in front of certain buildings and not to take off from water surfaces. Intelligent turns could be made by the ATSsim, forcing aircrafts not to collide with objects in the environment. Collision detection between aircrafts is already implemented in ATSsim[9], and it is much possible that this code could be modified to work with other types of objects as well. Another possibility is to enable aircrafts to find the closest exit to a runway.

In order to achieve this, the ATC AirportBuilder requires some functionality that is yet to be implemented as this implementation lay outside the scope of this thesis project. However a discussion on what should be implemented follows in this section.

As mentioned in section 4.1, a runway shape may be represented by a large field rather than pathway-shaped. Only in a context with other overlapping shapes can exact pathway information be gathered. This needs to be addressed

in the implementation. One way may be to allow the user to modify these shapefiles when an airport is created/modified. By giving the user the freedom to erase overlapping layers a runway shape can be transformed according to user preferences. This may be a useful feature as it gives the user some control. At the same time it adds another task to the user.

Another solution is to let the system handle these calculations with a simple algorithm: Erase all areas of a runway area that has an overlapping area of a different area type than a runway. The algorithm is clarified in the picture series below (Figure 9).

Figure 9: From left to right, a road area, a grass area, a road area from which road area overlapping with grass area has been deleted.

This algorithm could either be performed once on each road-like area type within the ATC AirportBuilder prototype, permanently transforming the area or at runtime within ATSsim each time the pathway of the area is needed. The advantage with the latter is increased flexibility if buildings, grass and other overlapping areas were to be added or removed from the area shape. At this time, having this extra flexibility seems unnecessary, and it comes at a cost of performance. The analysis of which alternative is the better lay outside the scope of this thesis project.

Once pathways are deduced from an original road-like area type it would be of use to find a line shaped path within the pathway that aircrafts could run along. This way the aircraft will know exactly where, on a wider path it should run and avoid zigzag driving patterns all over the roadway. Some research has been done in ways to achieve this. A skeleton, or medial axis (Figure 10) can be determined in any polygon shape. R. Edwards discusses several different algorithms for doing so.[26]

[26] Edwards, Robert, 2010,
Determining the Skeleton of a Simple Polygon in (Almost) Linear Time
http://home.comcast.NET/~maptools/Skeleton.pdf (Accessed 2013-04-20)

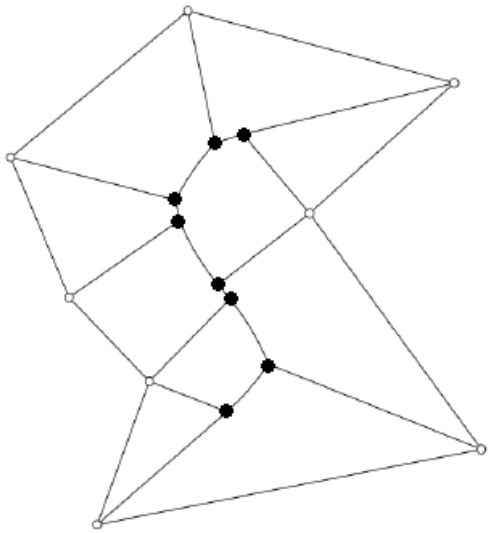


Figure 10: Medial axis

4.5.2 3D engine integration

Integration with the 3d engine would greatly reduce manual workload as 3d environments could be mapped from existing shapefile data. Some thought has been put into how this could be achieved.

The ATC AirportBuilder Wizard database has been given a table called area types. This class has been created with the ambition to take advantage of the information given in the .dbf shapefile. If the area type is given height, rendering/colour data is provided and the 3d mapping could be done without much manual work. (See section 4.1) This will however provide a 3d world where all objects of the same type look the same. The graphical experience of the product will therefore be less exciting than it is today. This is unless further algorithms are developed to enhance variety among objects of the same type. Creating more area types is another possible solution. Doing so requires that further attention is paid to creation and enforcement of .dbf shapefile standards.

5 Results

This thesis project has resulted in a prototype, ATC AirportBuilder, with an explanation of its functionality in section 5.3. The result section also discusses some future research possibilities in section 5.5.

5.1 Shapefile results

The shapefile was found to provide enough useful data for the creation of a prototype. Not only was spatial data provided, it was also found to include useful information in the .dbf files that may be used to draw basic conclusions on an objects height.

The data found in the Shapefiles is arranged as follows:

The .shp file includes all spatial references
The .dbf file includes miscellaneous data as attributes.
The .shx file includes indexing allowing for quick access of objects.[14]

The spatial reference data in the .shp was found to be essential to the ATC AirportBuilder, as ATC AirportBuilder revolve around area information about airports. However to suit the needs of the ATC AirportBuilder, .shp data needed some additional work (described in section 4.3.5) The attributes in the .dbf file were found to sometimes be useful as the attributes not always follow a set standard. When following the Falken standard (see section 1.3.2), height information may be deduced from the attribute "OBJTYP". Other miscellaneous data in the .dbf file may be used to provide the user of the prototype with information that he/she may find useful for manual data modifications.

5.2 Database model and data results

5.2.1 Tables

For structural reasons in this thesis, the tables needed to store shapefile data in the prototype, have been divided into two general groups. "Airport-related data" and "Area-related data." These tables are described below. A scheme of all tables and their relationships can be found in section 7.2.

5.2.1.1 Airport-related data tables

This table contains the common properties of an airport. The description of each field can be found in the list below:

Table 1, Airports

Field	Description
id	Unique id for the airport. Assigned by auto-increment.
name	Name of the airport.
airportCode	The ICAO code of the airport.
description	A description of the airport.
date	An automatic timestamp of when the airport is created.

This table serves as a link between table Airport and table Resource as the relationship between the two entities is of the type "many-to-many". The description of each field can be found in the list below:

Table 2, AirportsResources

Field	Description
id	A unique id for the airport the resource has a relationship with.

resourceId	A unique id for the resource the airport has a relationship with.
id	Unique id for an airport-resource relationship. Assigned by auto-increment

This table contains the common properties of an airport resource. The description of each field can be found in the list below:

Table 3, Resources

Field	Description
id	A unique id for the resource the airport has a relationship with. Assigned by auto-increment.
type	The type of the resource.
description	A description of the resource.

5.2.1.2 Area-related data tables

This table contains the common properties of an area. The description of each field can be found in the list below:

Table 4, Areas

Field	Description
id	A unique id for the area. Assigned by auto-increment.
name	Name of the area
description	A description of the area.
height	The height of the area.
areaType	The type of the area
airportId	The id of the airport to which the area is contained.

This table serves as a link between table Area and table ShapeType as the relationship between the two entities is of the type “many-to-many”. The description of each field can be found in the list below:

Table 5, Shapes

Field	Description
Id	Unique id for an area-shapeType relationship. Assigned by auto-increment
coordList	The spatial data of the shape.
shapeType	A unique id for the shapeType the area has a relationship with.
areaId	A unique id for the area the shapeType has a relationship with.

This table contains the common properties of an area shape type. The description of each field can be found in the list below:

Table 6, ShapeTypes

Field	Description
id	A unique id for the shapeType the area has a relationship with.
description	A description of the shapeType.

This table contains the common properties of an area type. These fields may be useful in future editions of the ATC AirportBuilder when integration with 3d engine is implemented. The description of each field can be found in the list below:

Table 7, AreaTypes

Field	Description
id	A unique id for the areaType.
Description	A description of the areaType
Texture	A texture for the areaType. Has been prepared for future integration with 3d engine.
Colour	A colour for the areaType. Has been prepared for future integration with 3d engine.

5.3 Implementation results and possible future work

This thesis project has resulted in the design and implementation of the ATC AirportBuilder Wizard that enables users to open and display shapefiles, modify useful simulation data and prepare this data for storage in a database. This wizard involves 3 or 4 steps depending on what data the shapefiles contain. This section will give a quick walkthrough of the most vital parts of the ATC AirportBuilder Wizard.

When the wizard is started, the main window is displayed. At this point in the wizard procedure, there is not much to do here as no shapefiles have been opened to be displayed or modified. A new airport can be created through the "File -> New" menu option (See Figure 11).

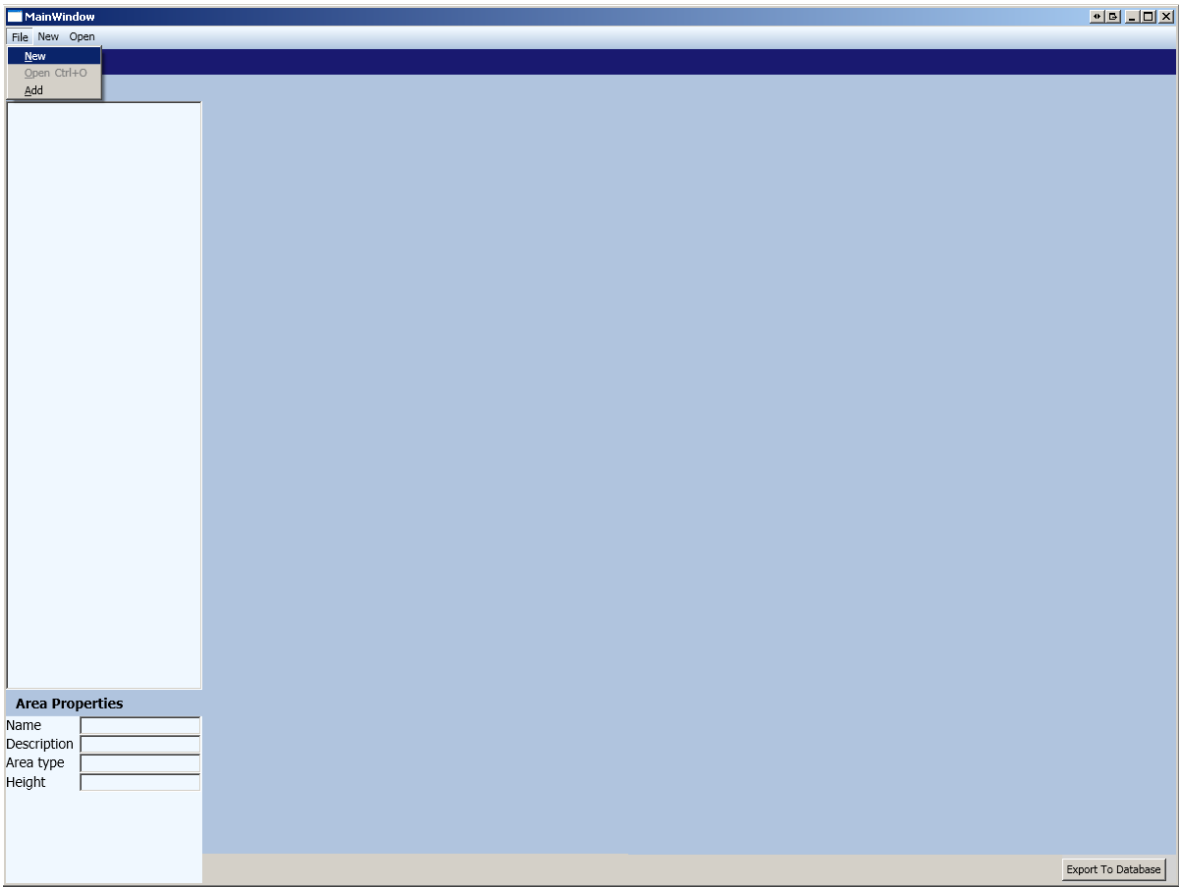


Figure 11:ATC AirportBuilder Wizard: window 1

The next step is entering the information needed to create a new airport object (See Figure 12). It is to this airport that the areas will be bound, as the airport is the central piece of information.

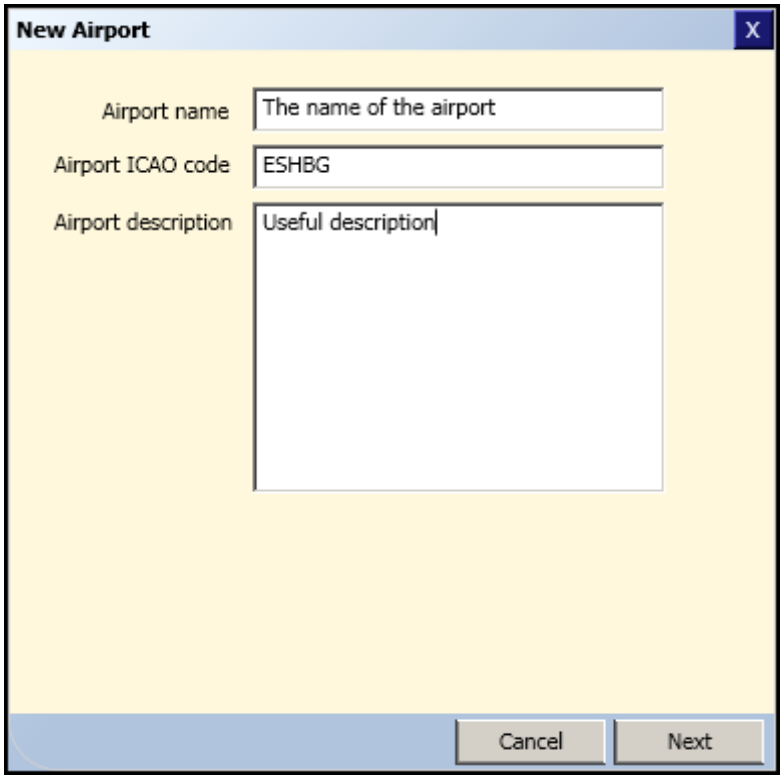


Figure 12: ATC AirportBuilder Wizard: New Airport Window

After the required airport information has been entered, a window (see Figure 13) where files can be selected will be displayed. The shapefiles that make up the airport environment are to be selected in this step.

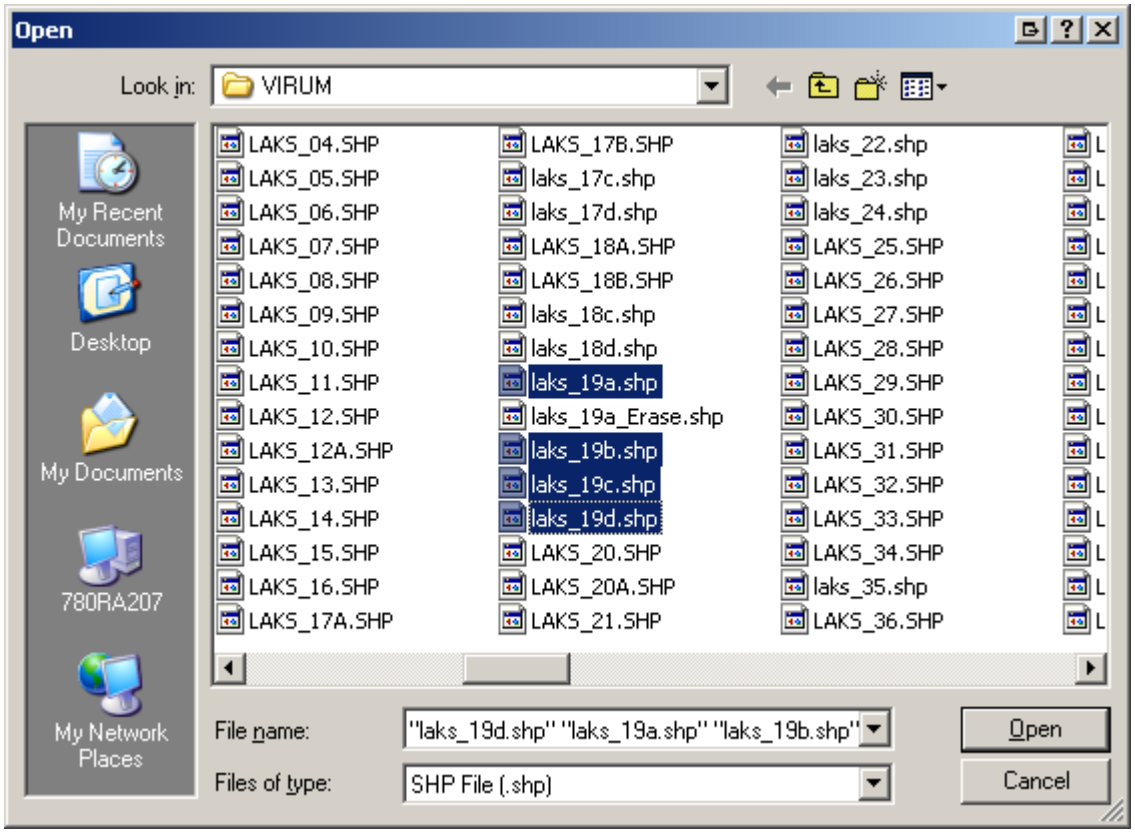


Figure 13: ATC AirportBuilder Wizard: select files.

The next window (see Figure 14) is not always displayed. It is displayed when some of the selected shapefiles do not contain any spatial reference system (see section 4.3.5). A spatial reference system must be selected for each of the shapefiles from the drop down menu, and to make the task less tedious, the user may choose to set a spatial reference system to all files that have the same value of a specified .dbf attribute. This is useful if the coordinate system is described by an attribute in the .dbf file.

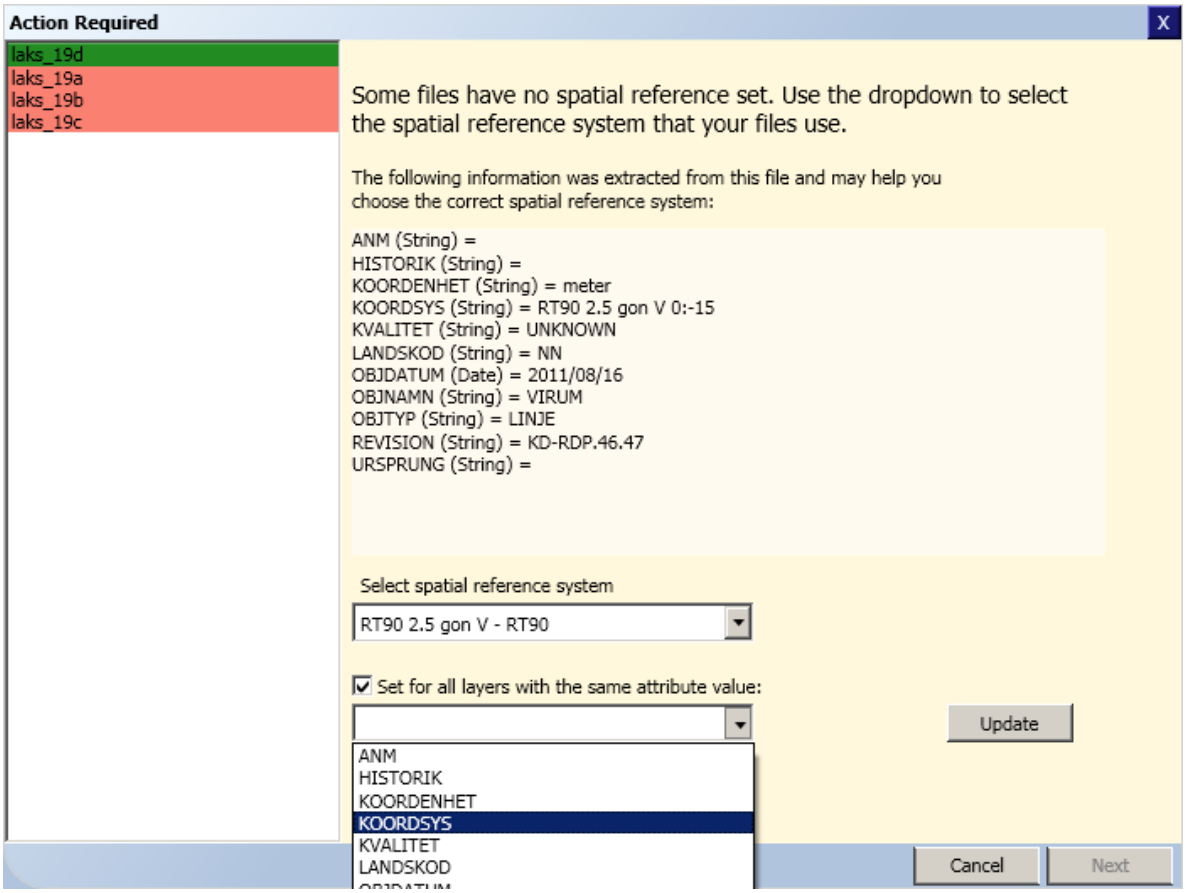


Figure 14: ATC AirportBuilder Wizard: spatial reference system.

Finally, the user is brought back to the main window (see Figure 15) again. Area and shape objects are created out of the shapefiles that were opened and are displayed on the screen. The user may modify the necessary attributes of each area and when ready, the airport, areas and shapes can be sent to the database for storage.

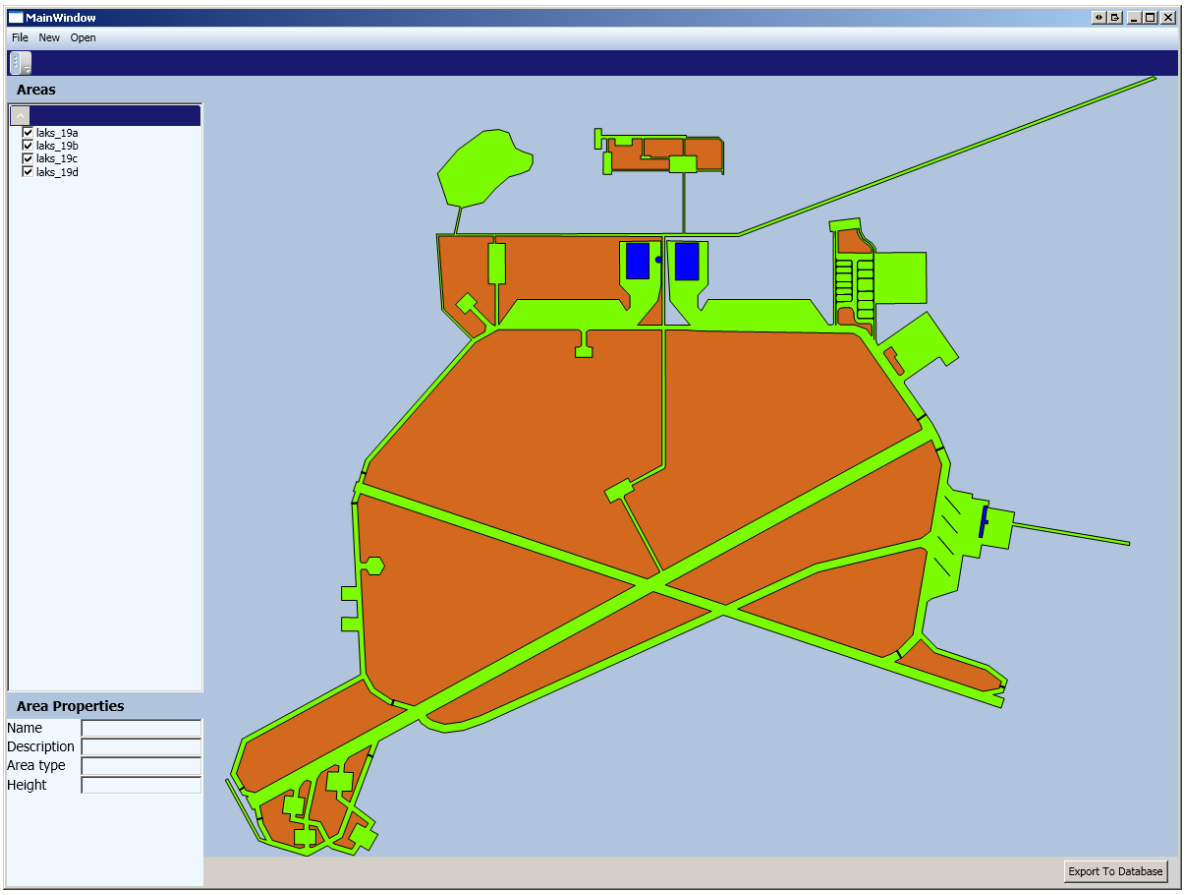


Figure 15: ATC AirportBuilder Wizard: Editor.

As the ATC AirportBuilder Wizard is yet a prototype, there are numerous possibilities to further extend the functionality of the ATC AirportBuilder in future versions. A few examples of desirable functionality follow:

- Automatic sorting of layers based on their area type
- Manual sorting of layers
- Graphical modifications of areas such as the ability to intersect, erase, merge, copy, paste areas among others
- Graphical selection of runways
- Save airport to file
- Apply skeleton algorithms in order to create pathways (see 4.5.1)

5.4 Performance results

The following statistics were collected during the performance analysis of SharpMap:

Method 1 VariableLayerCollection.TouchTimer(): 2-3% CPU and 37-38 Mb RAM
Method 2: Using MapBox.Refresh(): 3-4% CPU and 45-60 Mb RAM

The above results prove that SharpMap is fit for use in other systems as the load is low even on a relatively modest computer on which the analysis was performed.

The difference in CPU usage between the two methods is not significant, it is however expected to increase in more complex cases, as discussed in section 4.4, and the first method is clearly preferred. It is unknown to the authors why the second method consumes more RAM than the first, but the difference in RAM consumption is yet another reason to use the first method in future implementations.

5.5 Integration results and possible future work

In this thesis it has been found that integration with the ATC AirportBuilder and the ATSsim could be beneficial as it enables a broader range of ATSsim intelligence possibilities than the current system does. Some of these possibilities are as follows;

- Automatic plotting of taxi-routes and other pathways.
- Collision detection between aircrafts and non-aircraft objects in the environment.
- Intelligent aircraft collision-avoidance manoeuvres.
- Intelligent aircraft navigation.

In order to implement this integration the following points may be considered;

- Some areas in the ATC AirportBuilder Wizard database representing for example a “road-like” area, may be represented as a large field instead, when looked at in isolation. This problem can be tackled for example by shapefile data manipulation when a new airport is created. It is also possible to create algorithms within the ATSsim that performs area calculations at runtime.
- “Skeleton algorithms” (see section 4.5.1) could be used to help find pathways within road-like areas.

As well as there are benefits with ATSsim integration there are also benefits in integrating the ATC AirportBuilder with the 3dengine. Data from the ATC AirportBuilder could be used to map 3d environments thus greatly reducing the manual workload. In order to achieve this some points should be considered;

- Shapefile data is likely not to include height data. Instead, height data could be added to each area in the ATC AirportBuilder or area shape types could be used as reference. (See section 4.1).
- Shape types could be used to automatically give all areas of the same type a certain texture or colour.

6 Conclusion

In this section the questions asked in section 2 will be answered. There will also a few short conclusive notes on some of the most important results in the thesis.

An analysis revolving the first questions “What data is contained within a shapefile?” and “Which of this data can be used in the ATSsim and 3d engine?” can be found in section 4.1. There we have found that the shapefile does not only contain spatial data but also some miscellaneous attributes in the .dbf file. These attribute data may, if following for example the Falken standard be used to add basic height data to the spatial data from the .shp file. This data may then be useful when creating new airports in the 3d engine. There may also be use for this data in Scenario Management or in ATSsim where intelligence may be enhanced and scenario creation made easier.

The next question “What data tables will be needed in the prototype to store airport related shapefile data?” was discussed in section 4.2. From that section some conclusions have been drawn. Instead of including a field in the database with coordinate system format of an airport, the WGS84 system is always used. This not only simplifies the database table, it also ensures congruency with the WISE database. It was also decided to make use of the way “holes” are detected in an area in a shapefile that is documented in the ESRI shapefile documentation. There are existing algorithms that can determine holes from areas. The complete set of tables and their fields that have been chosen for the prototype can be found in section 5.2.1.1 and section 0.

Then there are a few questions that involve the implementation of the prototype. The question “How can shapefile data be viewed and modified?” is discussed in section 4.3. The analysis led us to an implementation in which some existing libraries were used. GDAL/OGR was used to extract data from shapefiles, perform calculations and prepare this data for database storage. SharpMap, a library based on GDAL/OGR, was used to draw spatial data on a screen. In order to avoid dependencies on the SharpMap technology, this library was used to the minimum. Those functions were isolated when possible. When answering the question “What sort of modifications may be necessary?” it was found that the spatial data needed some modifications. The shapes within each shapefile were separated, the shape names included before each new shape deleted and the coordinate system changed to WGS84. The wizard was also given some options in which the user could assign and modify shapefile data. For example areas can be given a new area type (grass, building etc.) and descriptions and names of areas may be changed. More about this can be found in section 4.3 and 5.3.

When the performance was tested in order to answer the question “How CPU-demanding is the method chosen for viewing shapefiles? “ it was concluded that SharpMap performed well. On the higher stress load where 400 points were to be rendered every hertz only between 2-4% of the CPU was needed of the test case computer. There were no indications found in that test case that SharpMap was not going to perform well enough for future versions of ATC AirportBuilder.

Lastly, there is the question on integration; “How can the extracted information from shapefile data be integrated with ATSsim?” In section 4.5 this question is discussed and some ideas are brought up. By making use of spatial data and area types (roads, builds etc.) intelligent algorithms to trace down roads could be made. Then possible taxi routes could automatically be suggested. The ATSsim could then also be given intelligence for finding the closest exit on a runway. Algorithms that determine medial axes on an area can then help find the correct pathway on an existing road.

A problem that needs to be dealt with those areas where their correct spatial information is only gathered when put in a context with other overlapping areas. (See section 4.1). How this problem can be solved is discussed in section. 4.5.1.

The ATSsim could, if given awareness of buildings and other “obstacle area types” be given some intelligence in collision detection. Perhaps the ATSsim could make an airplane automatically make a turn if about to collide with a building.

Then there is also the possibility of using the spatial data and basic height information gathered from shapefiles to automatically model the airport 3d environment. In addition, shapefiles can be used to automatically set textures and colours of items in the 3d environment. Lastly, shapefiles can be used to create the 3d environment itself, providing all 2d data. These options require some integration with the 3d engine.

7 Terminology

.dbf file	A shapefile file that contains attributes.
3d engine	Generates 3d environments to the ATC trainer TWR.
ArcMap	ESRI application that lets you open and view shapefiles
ArcCatalog	ESRI application that lets you open and view shapefiles
ATC	Air Traffic Control.
ATC AirportBuilder	The thesis project prototype.
ATC AirportBuilder Wizard	The thesis project prototype wizard.
ATC trainer TWR	Saab's simulated air traffic control tower for training purposes.
ATSsim	Saab's ATC training solution simulator
DBF Viewer plus	An application that lets you open .dbf shapefiles.
ESRI shapefile	ESRI's geospatial vector data format for geographic information system software.
Falken	A Swedish .dbf file standard developed by Saab.
GDAL/OGR	Open source libraries for raster and vector geospatial data formats.
GIS	Geographic information system.
IATS Client	Clients from which teachers and students may control airplanes throughout an exercise.
LINQ	Language Integrated Query.
MapBox	MapBox is a SharpMap control with built-in functionality for user interaction.
RT90	Swedish coordinate system.
SharpMap	An open source C# library for GIS data.
Thesis project	This thesis and all work involved.
VIRUM	Saab's fabricated airport data.
WGS84	The coordinate system used in Saabs ATC training solution
WISE	Widely Integrated System Environment. Saab technology that integrates various applications.
WKT	Well Known Text, a text markup language for representing vector geometry.
WPF	Windows Presentation Foundation.
XAML	Extensible Application Markup Language.

8 Appendix

8.1 Falken object types

Bilaga 5: Objekttyper för kartdata i FALKEN 1

Förklaring för typkoder:
L=Linje, S=symbol, T=Text, Y= Yta

OBJTYP	TYP	OBJTYP	TYP
AMA	L,T	RADARLEDNINGSSEKTOR	L,Y
ATZ	L,Y	RADARLEDNINGSLINJE	L
BASRESTRIKTIONSOMRÅDE	L,Y	RADARSTATION	S
BATALJONSGRÄNS	L	RAPPORTPUNKT	S
BORDERPUNKT	S	REC	S
BULLEROMRÅDE	L,Y		
CTA	L,Y	REFERENSNÄT	L,T
CTR	L,Y	REFERENSPUNKT	S
CTR_ÖVNING	L,Y	00_REVISION	T
D-OMRÅDE	L,Y	_REVISION	T
DB_RUTA	L,Y	REVISION	T
DELEGERAT LUFTRUM	L,Y	RIKSGRÄNS RGC/T	L
DELSEKTOR	L,Y	RIKSGRÄNS RGC/F	L
DME	S	RIKSGRÄNS	L
E1_BRYTPUNKT	S	ROUTE	L
EXEA	L,Y		
EXES	L,Y		
FARM	S	RWL	L
FIR	L	BANA	L,Y
FJÄLLOMRÅDE	L	RWL_RDP	S
FLYGHINDER	S		
FLYGKOMMANDOGRÄNS	L	S-OMRÅDE	L,Y
FLYGPLATS	S	SEGELFLYGSEKTOR	L,Y
FLYGPLATS_2KM	L	SEKTOR	L,Y
FLYGPLATS_6KM	L	SEKTOR_LINJE	L
		SEKTOR.STRECK	L
FLYGVÄGBEGRÄNSNING	L	SIGNIFIKANT PUNKT	S
FYR	S	SJÖ	L,Y
FÖRDEFINIERAT SPANINGSOMR	L,Y	SJÖ RGC/F	L,Y
HELIKOPTER-SEKTOR	L,Y	SJÖ RGC/T	L,Y
INFLYGNINGSLINJE	L	SPANINGSRUTA	S
INPASSERINGSPUNKT CTR	S	TERRITORIALGRÄNS RGC/T	L
KALIB	L	TERRITORIALGRÄNS RGC/F	L
KLP_RUTA	L,Y	TERRITORIALGRÄNS	L
KOMPANIGRÄNS	L	TESTTRANSPONDER	S,T
KORTBANA_LINJE	L	THR	S,T
KORTBANA_PUNKT	T	TI_JA37_FIKTIV_EJ_PRIO	S,T
KÄRNKRAFTVERK	S	TI_JA37_GRÄNS	S,T
L	S	TI_JA37_ORT_EJ_PRIO	S,T

LI	S	TI_JA37_ORT_PRIO	S,T
LO	S	TI_JA37_FYR_EJ_PRIO	S,T
LANDBRYTNINGSPUNKT	L,Y	TI_JA37_FIKTIV_PRIO	S,T
LANDKONTUR	L,Y	TI_JA37_FYR_PRIO	S,T
LANDKONTUR RGC/T	L,Y	TIA	L,Y
LANDKONTUR RGC/F	L,Y	TIZ	L,Y
LARMORT	S	TMA HUVUDOMRÅDE	L,Y
LÅGFLYGOMRÅDE	L,Y	TMA SEKTOR	L,Y
LÄNSGRÄNS	L,Y	TMA_ÖVNING	L,Y
MARKPUNKT	S	TÄTORT	L,Y
MÅL	S	UTA	L
MÅLOMRÅDE	L,Y	VOR	S
NATUROMRÅDE	L,Y	VOR/DME	S
NATUROMRÅDE_PUNKT	S	VÄG	L
NDB	S	VÄNTLÄGE	L,Y
ORT	S	VÄNTLÄGE_PUNKT	S
OSF-GRÄNS	L	ÄLV	L,Y
		Ö	L,Y
		ÖVNINGSSEKTOR	L,Y
R-OMRÅDE	L,Y	ÖVNINGSSEKTORGRUPPERING	L,Y

Figure 16: List of Object Types in Falken.

8.2 Data table relationships

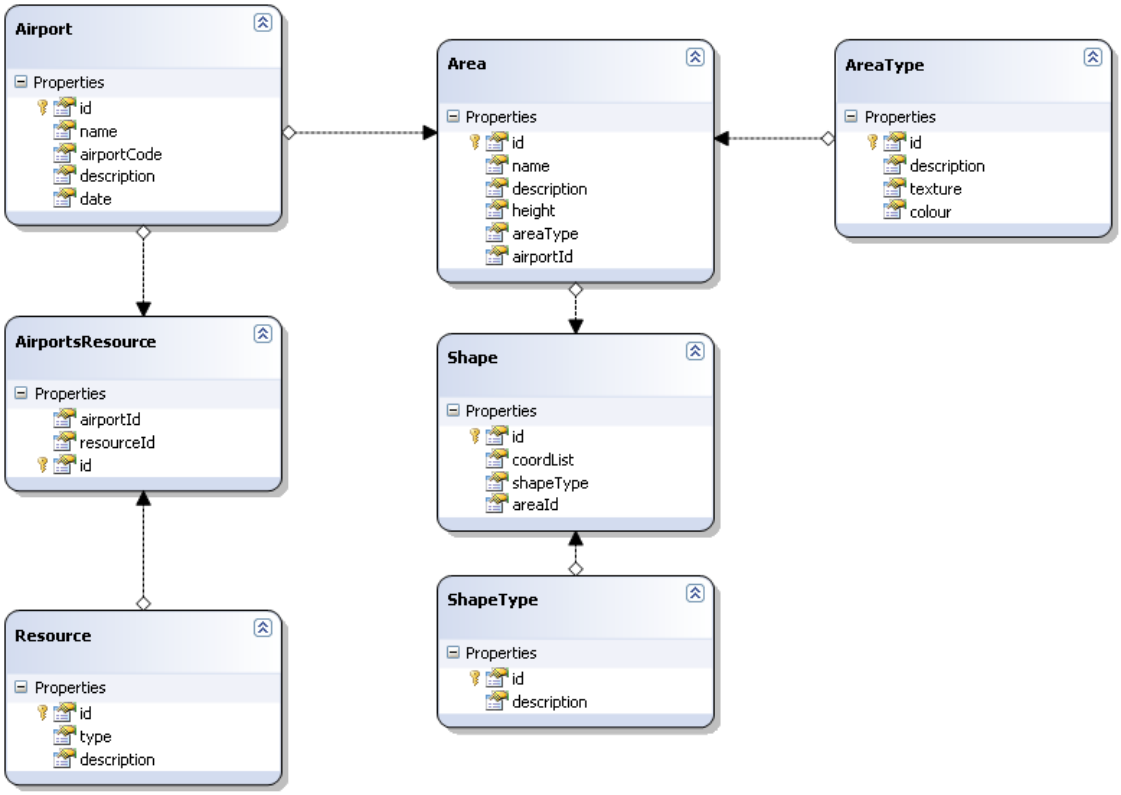


Figure 17: Data table relationships