CHALMERS

Modular Software Architecture for Connected Devices

The Development of a Model for Connecting a Device to the Android Platform

Bachelor of Science Thesis in Computer Science and Engineering

LOBNA ALHASSANI
MARIA SKÖLDIN GUSTAFSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, 2012

Abstract

The market is offering an increasing number of network connected devices and it is predicted that in the next decade a large number of electronic products will have network connectivity through which they will be monitored and maintained. Meanwhile the smartphone market is growing rapidly and the range of mobile applications provided is numerous. The challenge is to find a suitable modular way to connecting these products to PCs and mobile devices so that it is possible to access their information and to monitor/control them directly through these devices. Therefore the purpose of this research was to develop a software architectural model for connecting such devices, focusing on the modular characteristics of the architecture. An android application, Ship Detector, was developed with a connection to an AIS (Automatic Identification System) antenna that receives AIS signals from vessels and forwards them to the android application to be displayed to the mobile user. The study resulted in a model that could be used as a concept for developing similar applications. This model was demonstrated in a working prototype (Ship Detector) and presented with five diagram representations of the different perspectives of the application. Various challenges were encountered which are discussed further in the report.

Table of Contents

1 Introduction

1.1 Background

Today, we are witnessing a development whereby various types of electronic products are getting network connectivity allowing them to be monitored and managed through another, possibly remote device. According to Ericsson[1] it is foreseen that in the near future nearly 50 billion devices will have network connectivity and this will include devices such as kitchen appliances and other more complex machines.

The challenge we are facing in this development is how to connect these products to different types of computer devices and platforms in an efficient and simple way, a way that can be applicable to different types of devices.

Meanwhile, the consumption of smartphones and tablets has grown massively since smartphones were introduced to the mass market in the beginning of 2000. Smartphone sales have surpassed PC sales in recent years, nearly reaching double the sales (Gobry 2012) and according to a recent article on Business Insider[2] by 2016 it is expected that smartphones will account for two thirds of worldwide mobile phone sales.

Naturally we can expect an increasing demand for applications that facilitate a connection between mobile devices and other products. The convenience and efficiency with which a mobile device can obtain and control data that is generated by another remote product give this area great growth potential.

EIS Semcon specialises in system architecture, software and hardware design as well as electrical and electronic architecture. On the initiative of EIS Semcon, we will develop a general concept for connecting products to other PCs and mobile devices. This will be manifested in an application that obtains information from a connected product and present it to the user through a mobile device.

[1]http://www.ericsson.com/networkedsociety/media/hosting/Sustainable_Networked_Cities.pdf

[2] http://articles.businessinsider.com/2012-02-29/tech/31109577_1_smartphones-pc-sales-internet

1.2 Problem Statement

As the number of connected electronic products increases, more development is required to connect these products to PCs and mobile devices.

1.3 Purpose

The purpose of this research is to develop a software architectural design for connecting electronic products to PCs and mobile devices. There will be focus on the modularity of the architecture and the presentation of this model in the form of diagrams that describe the software design on different architectural and technical levels.

To demonstrate the architectural model, a server application will be developed that enables a mobile device to receive information from an antenna and display it to the mobile user.

The application in question will be developed for Semcon EIS to be used as a concept for future development of similar applications.

1.4 Delimitations of study

This study will focus on the architecture of such software. More specifically, the architectural design will revolve around certain characteristics such as modularity and modifiability.

There will also be a focus on developing a working prototype.

2 Method

The methodology of this study will include developing an architectural model based on previous work conducted by the Software Architecture group of the Software Engineering Institute (SEI) at Carnegie-Mellon University in Pittsburgh (Bass, Clements & Kazman 2007).

Moreover we will be developing an application based on this conceptual model using the programming language Java in the integrated development environment (IDE) Eclipse.

In addition, the application model will be presented in diagrams using UML (Unified Modelling language) with a diagram editor program.

3 Theoretical background

There are many aspects to this study and various concepts that are taken into account. Below is an introduction to these concepts and areas that make up the background for this research.

3.1 Mobile platforms - Android OS

Mobile phone devices became increasingly popular in the 1990s. Along with the introduction of the second generation (2G) of mobile platforms, mobile phones became a true mass market device. These mobile phone systems offered new forms of communication such as SMS (Short Message Service). The basic functions of making mobile phone calls and texting became widely used across all ages amongst the public.

However, in the late 1990s the vast majority of mobile phones had only basic phone features and as the use of mobile phones increased so did the demand for accessing the internet as well as other data services through mobile phones. The market was simply missing a mobile phone device and a platform that could account for all of these functions. In the long run it was also clear that 2G phones were not going to be able to meet the demand for greater data speeds and this is when the third generation (3G) mobile telecommunication was developed, to cater for such demands.

In the 2000s and onward, a new concept for mobile phones emerged, namely the “smartphone”. Despite the fact that this definition was used in the past, the concept of a smartphone was given a new meaning distinguishing it from the relatively basic “feature phone”. Although this distinction is vague, the most prominent difference is the supported advanced application programming interfaces (APIs) for running third-party applications (smartphone definition from PC Magazine Encyclopedia n.d.). Early releases of smartphones supported limited web browsing, as was seen with the first widespread smartphone in North America, Kyocera 6035 (Kyocera QCP 6035 Smartphone 2001). Later attempts included the Blackberry which became the first smartphone optimized for wireless email use. By December 2009, The Blackberry line had 32 million subscribers (2009).

But what truly changed the concept of smartphone was the release of the iPhone by Apple in 2007. Apart from its impressive hardware features such as the multi-touch capable user interface which started the industry trend for touch-only mobile devices, iPhone OS Software Development Kit (SDK) was introduced for Mac OS X, which allowed developers to create third-party native applications for the iPhone (Helal, Bose & Li 2012). This made the iPhone a mass market device whose capabilities could be modified, upgraded and customised using these third-party applications (Helal et al. 2012). The smartphone was no longer regarded as being exclusive to a certain privileged group of consumers or technical experts (Helal et al. 2012). Apple also released App Store, an application with a collection of all Apple certified applications developed by certified and uncertified developers. The concept of App Store became very popular as it offered an easier access, purchase and installation of iPhone applications.

Apple's iPhone was dominating the smartphone market for a while being regarded as far ahead of its competitors and offering innovative technology and services not available, or available to a lesser extent, by the other mobile phone and platform manufacturers.

The two other major native mobile platform competitors are Microsoft's Windows phone and Google's Android.

Android is developed and maintained by the Open Handset Alliance[3] which is led by Google. The Open Handset Alliance consists of many major mobile device manufacturers, chipset vendors, network operators and software companies. This includes Samsung, Motorola, HTC, Texas Instruments etc. (Helal et al. 2012).

Google acquired Android in 2005 and since its market launch in 2008 it has experienced explosive growth. Today, the Android platform runs on over 250 million devices and the number keeps increasing (Helal et al.2012). What sets Android apart from Apple's iPhone is that it is open source and can be developed and run on different devices by several manufacturers. In addition, Android is also used across a wide variety of mobile devices such as tablets, personal music players and in-flight entertainment systems.

Android allows anybody to download and customize its source code (except for the core platform) for their own use without paying any licensing fees or royalty. Additionally the Android Application Framework provides high level API (Application programming interface) and access to various functions of the platform for application development. Java is the programming language used for developing Android applications although

[3] http://www.openhandsetalliance.com

they are not fully compliant with the standard Java as there are significant differences in the user interface libraries.

The anatomy of an Android application can generally be built with 4 types of components namely: Activity, Service, Content Provider and Broadcast Receiver. In short, the Activity component (defined by the Activity class) represents a screen with a visual user interface and is while the Service component (defined by the Service class) is used for background tasks such as time intensive tasks or inter-application functionalities which do not require direct user interaction. The Content Provider component enables applications to store and share data with other applications and the Broadcast Receiver component (defined by the BroadcastReceiver class) is tasked with responding to system-wide broadcast announcements. Android can broadcast a number of system status messages such as device battery status or when the camera has just captured a picture (Helal et al. 2012).

In this study, we chose to develop an Android application and the choice of platform was based on its accessibility to developers and the variety of devices that it can be run on and in effect reaching a wider range of users.

The rapid evolution of the mobile phone, as a technical device but also as a mass market device conveys the great potential there is for continuing to develop mobile services and functions and in turn centralising the mobile device to becoming a device that caters for a wider range of demands.

3.2 Software architecture

A major focus of this study is to determine a suitable architecture for the application we are constructing and develop a design model that adheres to certain characteristics, specifically modularity and modifiability. The aim is to be able to apply the same conceptual model when developing applications with similar functions and to easily apply new functions to an existing application.

The term software architecture is a loose concept and has many definitions but there are still recurrent terms when describing it. According to Bass et al. (2007) software architecture of a program or a computing system is the structure or structures of the system, which comprises software elements, the externally visible properties of those elements and the relationships among them.

Extensive work has been done on defining software architecture and developing standards for good software design. A widely accepted definition is the one that came from a work done in the Software Architecture group of the Software Engineering Institute (SEI) at Carnegie-Mellon University in Pittsburgh (Bass et al. 2007).

As described by Bass et al. (2007) software architecture contains architectural patterns, reference models and reference architectures. These three concepts are suitable to use when describing software architecture although they are early decisions for software architecture and cannot by themselves be presented as architecture.

3.2.1 Architectural Pattern

An architectural pattern, or an architectural style, is a description of element and relation types together with a set of constraints on how they may be implemented (Bass et al. 2007). A common pattern is the Client-Server model which is also the pattern used in the application we are developing in this study. According to the definition of an architectural pattern, the server and client are described as the elements of a model while the set of constraints describe the way they communicate with each other according to a certain protocol (Bass et al. 2007). The Client-Server pattern is an informal definition but it is widely used and understood and mainly indicates that it contains several clients but no more than one server. The patterns are a very useful concept to use in software architecture but it is itself not an architecture (Bass et al. 2007). The choice of pattern is one of the first big decisions that an architect makes, and this is not chosen randomly, but is picked thoughtfully.

3.2.2 Reference Models

The second concept that Bass et al. (2007) explain makes up a software architecture is a reference model. A reference model is a tactic to solving a known problem by dividing the problem into smaller fragments or modules, each of which account for one part of the problem and cooperatively solve the problem as a whole.

3.2.3 Reference Architectures

The reference architecture is the third concept that describes software architecture (Bass et al. 2007). This concept is described as the mapping of a functionality (the reference model) onto the system decomposition (Bass et al. 2007).

The three concepts mentioned above sum up the notion of software architecture, but how important are these concepts when putting them in practice?

3.2.4 The Role of Architecture in Software Development

Nowadays computers can be found in many different environments. More and more machines and various technical products contain some sort of a computer, whether it is visible or embedded in a larger system. Along with the computer there is also software that takes care of the data input and serves a function. As these systems become larger and more complex, software architecture becomes more important as it needs to account for more complex challenges and provide a software model that eventually can be tested, rebuilt, updated and maintained (Rozanski & Woods 2005). The architecture also needs to facilitate a coherent software model that stakeholders can take part of and understand.

According to Bass et al. (2007) there are three main reasons as to why software architecture is important. The first one concerns the communication with and among stakeholders. All stakeholders have their own agenda with different qualities they want to impose on the system, and it’s partly up to the architect to consider the stakeholders demands and make it all work together. The presence of good software architecture also enables different stakeholders to work separately on different parts of the project while ultimately working towards the same goal. Another important factor is that architecture facilitates early decision making regarding a system. These early decisions are the ones that define the system and make the foremost impact and are the hardest to change later on in the system development process (Bass et al. 2007). Bass et al. (2007) also emphasize the importance of software architecture in terms of the possibility to reuse other design decisions that have been previously built and tested and proven to work well. In addition, reusing older design methods also saves time as an architect do not need to develop methods that may already exist.

3.2.5 Modularity

Modularity in software architecture refers to a cluster-based architectural style by which different clusters, or components, are connected together. These components comprise of well-defined functions which together serve a common function. Modular characteristics facilitate easier addition and modification of functions within a system without greatly impacting other parts of the system. Modularity is a part of a broader concept in architecture, namely modifiability.

In system architecture, the attribute modifiability refers to how costly it is to make a change in a software system (Bass et al. 2007). This type of change does not refer to the changes that end users and system administrators can make but rather to the functional changes, such as adding a new feature or other changes to the source code.

Typically, before such changes can be applied to an existing system, all necessary specifications must be drawn up and designed to fit into the existing architecture while at the same time meet the functional requirements. In addition, the new implementation must be tested and deployed. Thus, making a functional change to a system requires a lot of work and can become very time consuming and costly if the system is designed in a way which would require modification in many other parts of the system (Bass et al. 2007). This scenario refers to the ripple effect which is when modifications must be enforced on a module that is not directly affected of the change (Bass et al. 2007). For example if a feature is added and some changes must be implemented in module A, a ripple effect on module B is when alterations must be applied to this module only because of the changes in module A and not because of the new feature (Bass et al. 2007).

As such the architecture of a system should be structured in a way that allows cost efficient changes. This can be accomplished by minimizing the number of modules affected by the changes restricting them to a small set of modules.

The Importance of Good Documentation

When an architect constructs an architectural model for a software, there are many aspects and perspectives that are taken into account as well as different architectural levels that convey different degrees of detail. In order to explain the model correctly to technical and non-technical stakeholders good documentation needs to be provided. To have records of a system that all stakeholders can read and interpret is very important (Rozanski et al. 2005). When groups of specialists are working on different parts of the same project separately the documentation is the guide for all stakeholders to be able to aim in the same direction. A good way to represent the architecture is with diagrams and pictures that will show different degrees of depth. Not all stakeholders are interested or can even understand very detailed diagrams of the system and so perhaps it would be suitable to provide an overview of the system that conveys the information they need (Rozanski et al. 2005). For example if a salesperson is going to present a software concept to a new client, there is no point in showing very detailed presentations of technical terms and concepts to the client that might not even be a technical person. The technical presentations may be of interest later on in the sales process when IT professionals get involved.

Today there are some well-known standardised graphic representations of different architectural concepts. Different components of a software model can be represented by different symbols and arrows that correctly convey the design decisions without having to add extensive descriptions in text.

3.3 Agile Software Development

A prerequisite for developing our application was to work in an agile manner, a developing method which has been widely used in recent years in the software development practices. The information regarding agile software development in this section is based on the principles of the Manifest for Agile Development[4].

Agile software development refers to a set of software development strategies that are based on iterative and incremental development. During the development process, requirements and solutions evolve through the collaboration between self-organizing and cross-functional teams.

The principles of the agile development methods are defined by Manifesto for Agile Software Development, a document that emerged in 2001 when seventeen software developers were looking for a better way to develop software. This group of representatives with experience from different developing disciplines were interested in a new way of developing software instead of the former traditional documentation driven, heavyweight software development as they described.

The characteristics of an agile development process specify that in the beginning of a new project not all requirements are stacked out with a detailed software architecture. In an agile process the software is developed in a number of sprints which in this context refers to a period of time by which certain amount of work has been performed. On each sprint meeting the requirements for the upcoming sprint is outlined and further executed during the sprint. Hence after every sprint there should be an existing software with the specified additional requirements set out on the last sprint. This way the product will grow as the client gives new requirements

[4] http://agilemanifesto.org/iso/en/

in the beginning of every sprint and can see a working software at the end of every sprint. This enables the client to have more control over the development process and can decide which requirements to be prioritised etc. Experience has shown that this way of software development is preferred by clients.

Agile software development has further been branched into various agile methods each of which has certain characteristics that may be more suitable for different developers and different types of software projects. One of those agile methods is called Scrum and is highly used in the area of software development.

Scrum

The following section will describe the principles of Scrum and the procedure when choosing to work with Scrum. The information in this section is based on the Scrum documentation by Schwaber & Sutherland (2011). Full references are provided in the reference section.

Scrum is built on the principles of agile development. In Scrum the team involved consists of three roles; product owner, Scrum master and the development team. This team decides how the work will be conducted and this is done without the involvement of an outside management.

The development team adopts a flat structure where everyone is regarded as a developer, possibly with different skills and expertise but with no specific hierarchical titles.

The product owner is responsible for the quality of the development team's work and the product backlog, which is a list of all requirements that should be implemented in the software. Furthermore, the product owner ensures that all requirements are clearly defined, understood and implemented in the right order by the developers.

The Scrum master ensures that the principles of Scrum are understood theoretically and practically by everyone in the team and that these principles are followed through. The master also and looks at the efficacy and provides support to the development team.

On a more detailed scale, a project that adopts the Scrum developing method is initiated through a sprint planning meeting. This meeting is the first event of a sprint and decides the workload of the following sprint. This meeting discusses two central issues, what to be done and how it will be conducted. The product owner present items, or tasks, from the Product

Backlog which are ranked according to which is the most important at this time. The developers decide how many items to include in the upcoming sprint. The whole Scrum team collaborates around the work that will be executed. Once the number of tasks is selected, the team specifies how the selected items will compile into the product.

Daily Scrum
In addition to the sprint meeting, there are short meetings held on a daily basis by the Scrum master. The team decides what will be done during the day, until the next short meeting. During this meeting the developers go through what was accomplished since the last meeting, what will be done for to the next as well as the potential obstacles that can occur.

Sprint Review
In the end of each sprint a sprint review meeting is held to discuss how the process of the sprint went. The product owner is briefed on what has been done and a demonstration of the product is made. The development team address the problems they encountered and discuss future workload etc. During this meeting important points are discussed that are brought along to the next sprint planning meeting.

Sprint Retrospective
This meeting is set between the sprint review and the sprint planning meeting and it addresses the previous sprint with regards to the group dynamics, in terms of how the team worked together. At this stage a plan is made up on how to improve the next sprint in terms of how to make the group of developers more efficient and make it more enjoyable for all members in the Scrum team.

3.4 Automatic Identification System (AIS)

The information in this section is based on the information provided by the Swedish Maritime Administration. Full references are provided in the reference section.

In our study, we are developing a mobile application that will connect to another device and make use of its data in various ways. As such, this device will be a local antenna that receives so called AIS (Automatic Identification System) signals. This section will provide a brief explanation of AIS signals, the meaning of and the concept behind it.

AIS is a system which enables vessels and land based stations to receive information about other marine vessels and is considered a compliment to the radar system. The idea behind the AIS system is to provide means for collision detection among large vessels at sea that are not within the range of shore-based systems. However, it only determines the risk of collision and is not used as an automatic collision avoidance system.

AIS signals that are sent from a specific vessel include information about the vessel such as its unique identity number (MMSI), navigation status, size, destination, position in longitude and latitude etc. More details about the content can be found in Appendix 1.

AIS signals use VHF (Very High Frequency) radio channels and send their content in small data packages. These signals can be received by vessels that have AIS equipment and are within the range of the VHF radio transmission. The AIS signals are sent in intervals where signals with information that frequently changes such as position, course and speed are sent every 2-10 seconds and information like size, type of cargo and destination are sent less frequently.

Since 2002, vessels that follow the SOLAS (Safety of Life at Sea) convention[5] and are bigger than 300 tonnes are obliged to implement the AIS system. In 2007, all vessels trafficking national waters and all passenger boats in international waters, irrespective of size, implemented the system.

3.5 Application Programming Interface (API)

An application programming interface (API) is a source code based specification which is used as an interface for the communication between software components (Orenstein D. (2000)). An API specification can take many forms such as a library for a programming language (for example Java API) or in the context of web development, a set of HyperText Transfer Protocol (HTTP) request messages along with response messages formatted in an Extensible markup Language (XML) or JavaScript Object Notation (JSON). Typically, the http responses and requests vary in content depending on the type of information being requested as well as which web service is being contacted. However there are some standards that are

[5] "The main objective of the SOLAS Convention is to specify minimum standards for the construction, equipment and operation of ships, compatible with their safety." *International Convention for the Safety of Life at Sea (SOLAS), 1974* n.d.

required in most web service requests. Usually, a request should contain a key that is issued by the API provider. Most keys are free and easy to obtain, and they enable the API provider to keep track of the number of API users and who is using it. Another typical request specification is which format the response should be in (e.g. XML).

Web APIs allow the combination of multiple services into new applications known as mashups. These mashups can generate new information by combining two or more sources which originally was not provided by either source.

It is common that Web communities share content and data with other communities and applications. This content can be dynamically posted and updated in multiple locations on the web and this is evident on for example social networks. The Facebook API is often used on other sites, such as media websites whereby web users can interact with the content of the site by using their Facebook accounts. Another example is sharing video content which is embedded on sites served by another host. This is seen in applications that implement the YouTube video framework to post videos.

Creating an application that gathers information from a connected device and combining this information with one or two open API sources can produce unique information that can be useful for private as well as professional use.

3.6 Cloud Services

Generally, the IT security within companies is very strict limiting incoming and outgoing communication. This is no exception to the IT security on Semcon. While this is necessary for network protection, it limits developers when developing systems that need to freely communicate over the internet. For this purpose a cloud service was eventually used for our application to allow such communication without compromising the security risks.

Cloud services offer storage space and computing on a remote device that can be accessed remotely from another computer device. Cloud services are becoming increasingly popular amongst private users and companies.

This type of service is typically offered by a company that takes care of the maintenance of the physical servers as well as network connectivity. Clients can choose between a range of virtual private servers (VPS) with different operating systems, different amounts of storage capacity and other relevant features. Even if the server belongs to the provider, the user holds administrative rights and decides e.g. which programs to install.

The user connects to the VPS through an application that establishes remote connections. For example, on Windows the user can connect to the VPS through a program called Remote Desktop Connection where the user is able to view the graphical interface of the remote server. Telnetting the VPS is also possible.

Cloud services can optimise the usage of storage space where users can share hardware and take up the storage space they need. This is advantageous from an economical as well as an environmental point of view.

4 Our study

4.1 Goal

To go back to the purpose of this report, the idea was to develop a software architectural design for connecting product devices to a mobile platform (or other types of PCs) focusing on modular design characteristics. This model was going to be presented in a set of diagrams that describe the software design on different architectural and technical levels. Additionally, to demonstrate the architectural model, a software application named Ship Detector, would be developed that integrates the above features.

Ship Detector is described in detail in the following sections.

4.2 Introduction to Ship Detector

The idea of the application Ship Detector is to interpret AIS data taken from an AIS Encoder and display this vessel information to the end user by mapping out the vessels on a map on a mobile device.

Ship Detector consists of a server application that runs on a java supported device and a client application that runs on the Android platform. The server application receives AIS signals as raw data strings from an AIS Encoder that in turn fetches these signals from a local antenna installed on the Semcon Building. The server on Ship Detector can thus access, interpret and store information about adjacent vessels. The client application contacts the server for vessel information and stores it in its local database and then displays a representation of the vessels on a map on the android device as shown below in Figure 1.

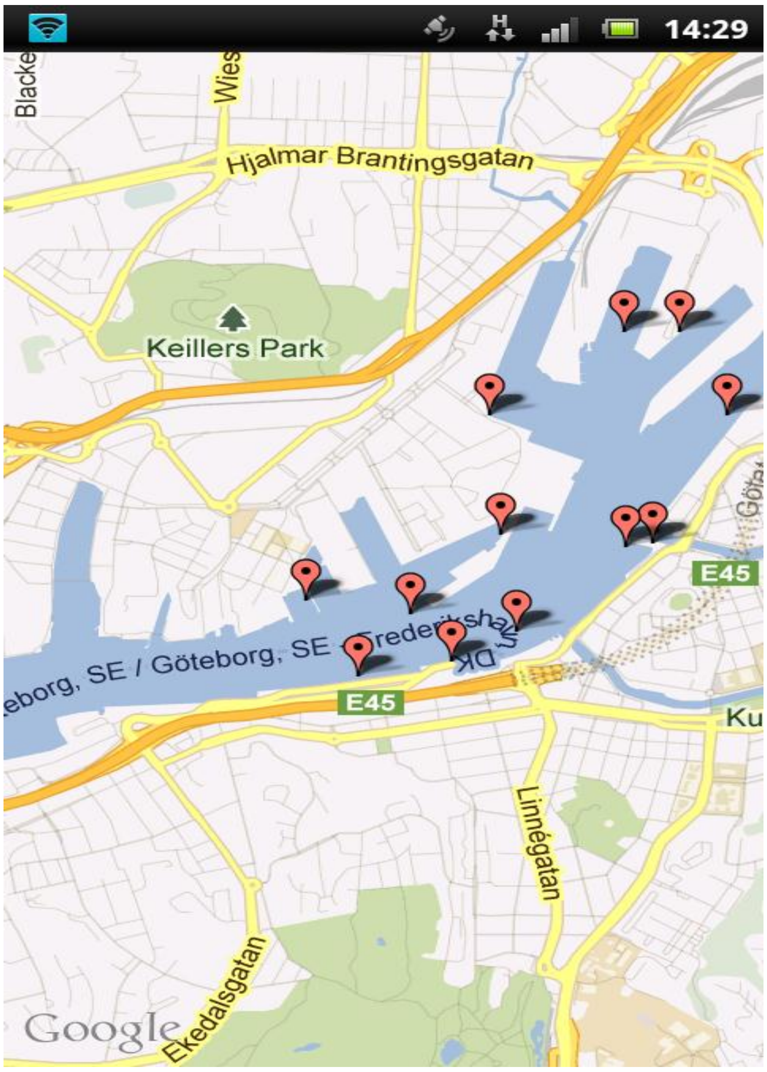
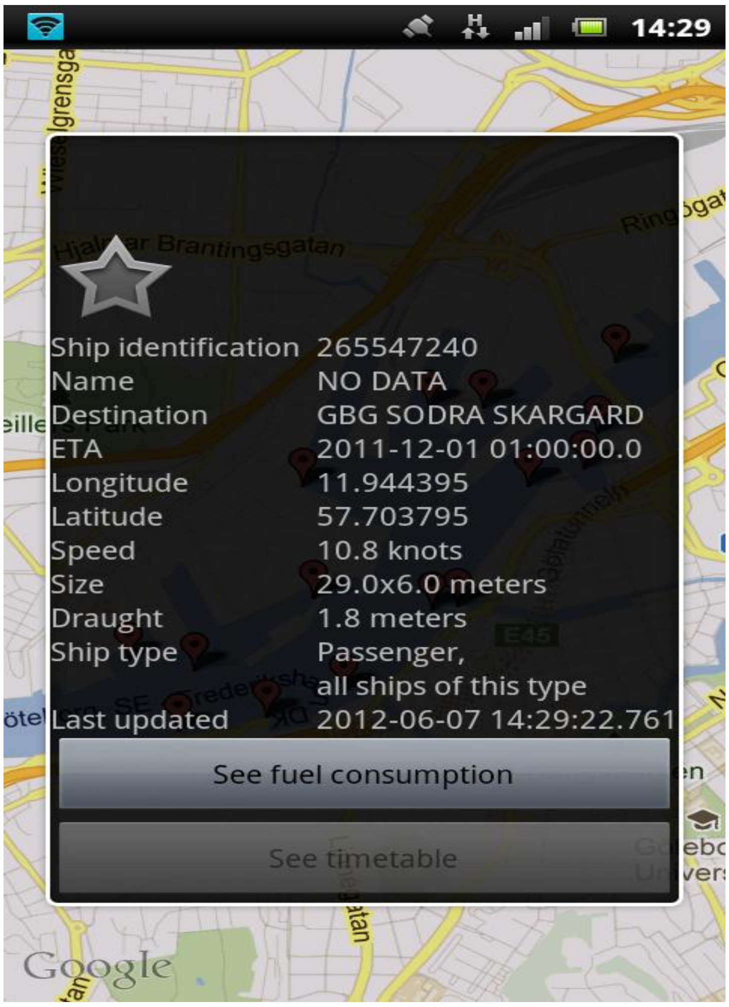


Figure 1 showing markers representing vessels.

When vessels are displayed the user is able to tap on any vessel marker and a popup view will be shown with vessel specific information as shown in Figure 2.

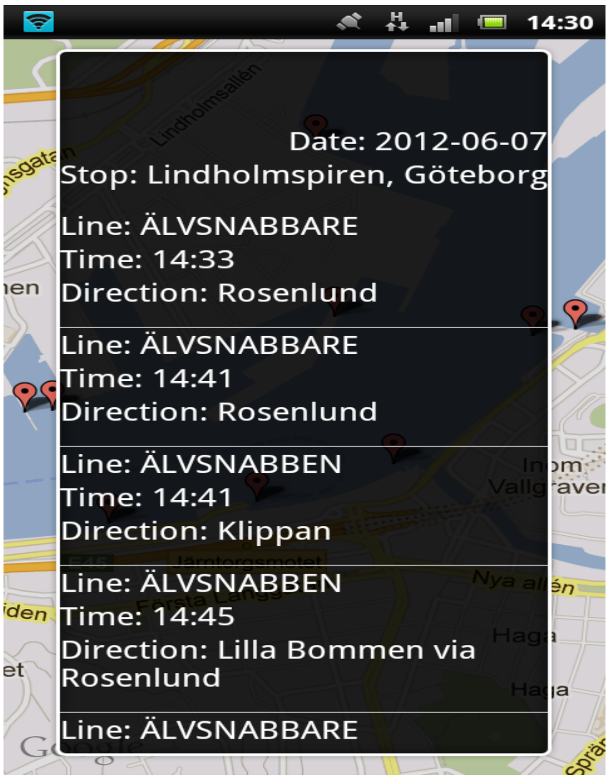


Ship identification	265547240
Name	NO DATA
Destination	GBG SODRA SKARGARD
ETA	2011-12-01 01:00:00.0
Longitude	11.944395
Latitude	57.703795
Speed	10.8 knots
Size	29.0x6.0 meters
Draught	1.8 meters
Ship type	Passenger, all ships of this type
Last updated	2012-06-07 14:29:22.761

Figure 2 showing a popup window displaying information about a specific vessel.

The local database on the Android device is updated periodically with the help of a timer that sends a request to the server for new information.

Additionally, Ship Detector makes use of three APIs, Västtrafik, CommuteGreener and Google Maps which enable the user to view the timetable for urban transport vessels as well as calculate CO2 emissions for intercity journeys. The user can also mark specific boats as “favourite boats” to be able to track them. These features are shown in Figure 3, Figure 4 and Figure 5 below.



Date: 2012-06-07
Stop: Lindholmspiren, Göteborg

Line: ÄLVSNABBARE
Time: 14:33
Direction: Rosenlund

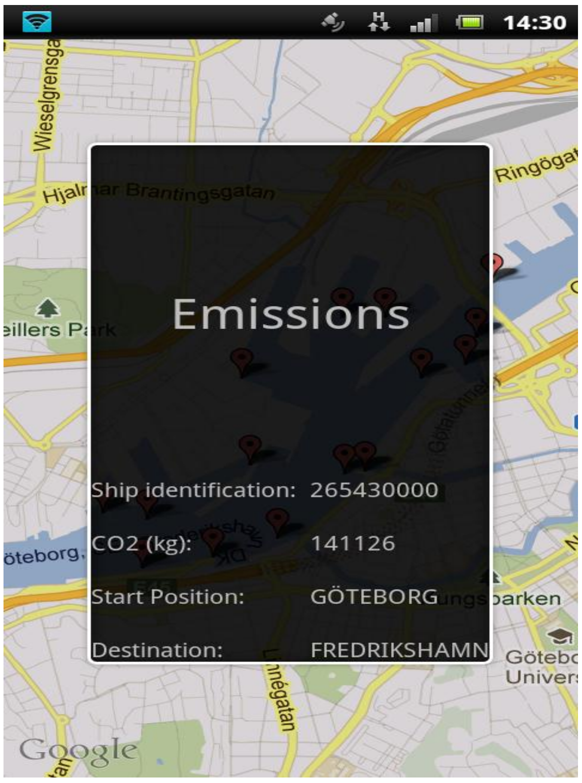
Line: ÄLVSNABBARE
Time: 14:41
Direction: Rosenlund

Line: ÄLVSNABBEN
Time: 14:41
Direction: Klippan

Line: ÄLVSNABBEN
Time: 14:45
Direction: Lilla Bommen via Rosenlund

Line: ÄLVSNABBARE

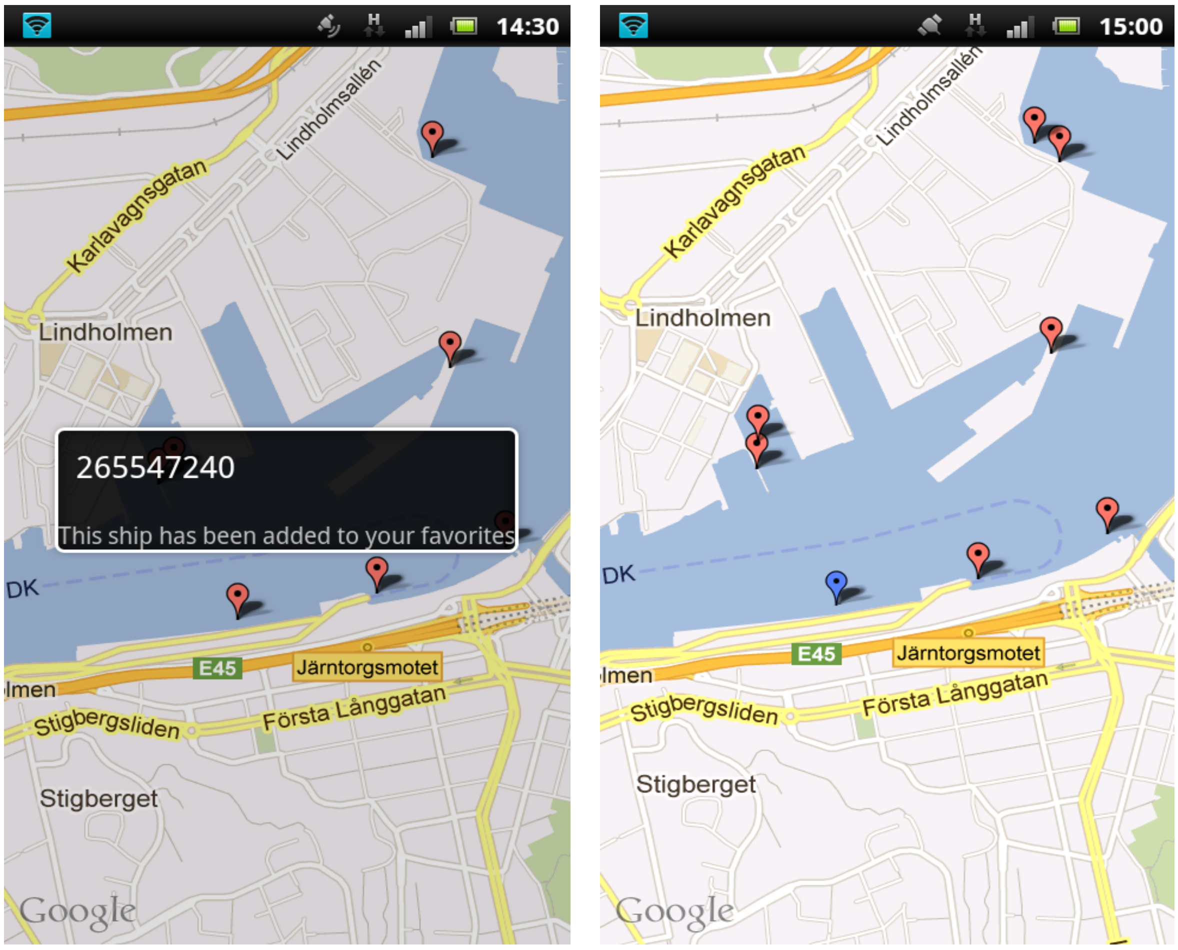
Figure 3 showing the timetable for the nearest stop for public transport vessels. This information is taken from the Västtrafik API.



Emissions

Ship identification: 265430000
CO2 (kg): 141126
Start Position: GÖTEBORG
Destination: FREDRIKSHAMN

Figure 4 showing the CO2 emissions for a vessel going from Gothenburg to Fredrikshamn. This information is taken from the CommuteGreener API and Google Maps API.


265547240

This ship has been added to your favorites

Figure 5: A user can mark a vessel as “Favourite”. The marker for the vessel will get a different colour.

4.3 Methodology and technologies

4.3.1 Development Environment

The application was developed with the programming language Java in the integrated development environment (IDE) Eclipse. Google code was used as a subversioning tool and for simultaneous development of the application amongst the group members. Google code was used with the help of a plugin to Eclipse, Subclipse, which enabled the group members to commit and download source code from a project on Google code.

In addition, to be able to develop an Android application, several additions were used which were added to Eclipse including Android Development Tools (ADT), Software Development Kit (SDK) and Android Virtual Device (AVD) which is an emulator that simulates an android device.

Development method
The application Ship-Detector has been developed in an agile manner partly using Scrum. To reference back to the Scrum method it was not possible to use it at its full potential. During the development process there was no real product owner, the Scrum master and product owner was the same person (supervisor at Semcon) and the requirements for the application were put forward from both the product owner and the group members, the development team. Weekly sprint planning was held in which the product owner and the development team staked out what had be done during the upcoming week. For every function or requirement to be implemented a "ticket" was made which is a task or a user story to be implemented along with an estimated duration. In addition, the development team held daily Scrum meetings to plan the daily workload and wrote a diary highlighting the daily work along with occurring problems and references to useful links.

Automatic Identification System (AIS)
The functionality of Ship Detector relies on AIS signals. These signals are received by an antenna located on the Semcon building in Lindholmen, Gothenburg. These signals are further handled by an application, AIS Encoder, which is placed in a Linux based server. The application receives AIS signals in binary code and converts the data to strings of characters that are sent out to connected clients. The Ship-Detector server connects to the AIS Encoder and receives the strings of raw data. At this point, the strings of data do not convey any useful information and must be further converted. To allow this conversion, Ship-Detector has integrated a free converting application from freeais.org that converts the raw AIS data to Java objects.

4.3.2 Application Programming Interface (API)
Ship Detector makes use of APIs to obtain additional information about specific vessels. To obtain this information http requests are sent with a string of characters that contain information about the address of the web service and the type of information requested. The response of the request is received as a string of characters in the structure of JSON. In order to convert this data, the server application makes use of a JSON library that converts the string of data to a Java object. The information is then easily retrieved.

The APIs used in Ship Detector include Google Maps, Västtrafik and CommuteGreener. Google Maps is used to obtain coordinates of cities and addresses by sending a request with the name of that city or address.

Västtrafik is used to obtain the timetable for a few specific stops around Götaälv where public transport vessels stop. As for CommuteGreener, a calculation of CO2 emissions is obtained for a specific vessel based on the start and end coordinates that are sent in the request. It is worth mentioning that CommuteGreener only covers urban transport vessels.

4.4 Software architecture of Ship Detector

4.4.1 Application Model

The system architecture has focused on separating the application into divisions, each of which has consistent and common responsibilities. Ship Detector adopts the client server model. The client is the source that establishes a connection with the server and the server in turn responds back to the client. The server functions as a passive server, being the source that only responds to the clients. The server allows multi-client connections.

The client adopts a three-tier structure while the server has two tiers. The client has a graphical user interface, a logic tier and a database tier whilst the server has a logic tier and a database tier.

In the client- and server application there are two different databases and these vary in function and structure. The server database must be available at all time as it functions as the central database which stores all the vessel information for long periods and is accessible to enquiring clients. The database on the client side stores information about the vessels locally on the mobile device and is updated regularly.

Client

The client adopts a three-tier structure to separate the graphic from the logic and the logic from the data layer. The separation between the graphic layer and logic layer facilitates structure and enhances modifiability whereby changes in one layer do not affect all other layers.

Presentation tier

Clearly, the graphic layer is the part of the application that accounts for the graphical user interaction, translating functions and results to the end user. This part has been developed for the Android platform. Despite of the efforts of trying to separate the presentation layer from the rest of the application many of the essential features of the application, such as Google map are dependent on the android related source code and so important features of the client application are located in the presentation layer.

Logic tier

All information from and to the graphical layer goes through a single point, the logic layer, and it is concerned with either collecting AIS data from the local database or establishing a connection to the server in order to acquire other information either stored in the server database or collected from an API (Västtrafik or CommuteGreener).

Data tier

The data layer handles the storage of AIS data on the local android device using SQLite. SQLite is the only database that can be used. It is possible to write data on a file and then store this on the device but then the search possibilities are very limited. The database stores information about vessels taken from the server database on the server. This database is updated from the server every few seconds. For the purposes of this particular application, perhaps a local database was not necessary to implement. The initial motivation for using a local database is to allow a user to access vessel information without having network connection even if this information would not be up to date. Currently, there are three tables of data stored, "Ships" which contain all vessel related information, "Positions" that stores the 10 latest positions obtained for a specific vessel and "Favourites" which contain the vessels that the user has marked as "favourite". More details on the client database can be found in Appendix 3.

Server

The server is divided into two layers, a logic layer and a data layer.

Logic tier

The logic layer is concerned with communicating with the external applications as well as the clients and handling various tasks related to these applications. Currently there are three rather separate parts within the logic layer, the "AIS handler", the "API handler", and the "Client handler". The AIS handler handles the reception of AIS signals, the conversion of these signals into objects, and the storage of these signals in the database. The API handler is concerned with the communication with the external web services, Västtrafik, Google Maps and CommuteGreener. It receives data in JSON format and converts it to a JSONobject which can be interpreted in java. The Client handler communicates with the client and calls the other handlers to acquire information to be sent back to the client. Another part to the logic layer handles the database logic.

Data tier

This layer accounts for the persistent storage of the AIS objects. A relational database is implemented using Apache Derby 10.8.2.2 with a Derby Embedded JDBC Driver. There are three tables, “Ships”, ”Positions” and “Destinations”. These tables are regularly updated as AIS data is received. The database handler makes use of SQL to simply set and get information from the database. There is no additional logic to this layer and the idea is to keep this tier as simple as possible and with a defined functionality. This makes transitions to other database types easier. There is only on point of connection to the database from the rest of the application.
More details on the server database can be found in Appendix 2.

4.4.2 Server/Client Communication

The client/server communication is conducted in a uniform manner and is object-oriented. A threaded connection is implemented to allow multiple clients to server communication. There is also a restriction of a number of simultaneous client connections. This is set to avoid performance decline and also to avoid external attacks which can lead to server overload.

Since the data sent between the client and the server can take many forms we had to design a system of communication that would allow us to send strings, integers and a range of different objects. To achieve this we created a system where objects sent inherited a class we named “Message”.

As described in this section, the design of Ship Detector is based on a layered structure which enhances the modularity of the application. For instance, the ripple effect which was discussed in the previous chapter is taken into account by restricting the communication between the layers to single points and thus calls for a function does not come from many different directions. For example, in Ship Detector, the path of communication is restricted as seen in the design of the client application. All calls from the graphical interface go through one point to the logic layer, before continuing to the data layer. On both the server and the client all communication with the database goes through a single point etc.

In addition, a number of methods and information within each module has been implemented as private to be able to hide as much information as possible so that changes can be isolated within each module and prevented from spreading out to other modules. This ensures that implemented changes would only affect the specific module that holds the function.

4.5 Presentation (Diagram)

One of the many challenges of software architecture is to present the software model and the vision behind it in a way that is understandable to both technical and non-technical stakeholders. Different stakeholders have different perspectives, requirements, levels of information and different treatment of information (Bass, Clements & Kazman 2007). Accordingly, five different diagrams have been drawn to present the relevant characteristics of the software model to the stakeholders involved.

Domain-level diagram

The domain level diagram is presented below. It is a high level diagram that is intended mainly for non-technical stakeholders and is used for explaining the basic idea behind the application. As shown, the diagram simply presents the main concept of the application which is the use of a server application that many clients (mobile devices in our case) can connect to through an internet connection. The server application is connected to an application, AIS Encoder which is a receiver for the AIS antenna. The server has a storage component that stores information that is acquires from AIS Encoder. The server is also connected to other information sources. The server is comprised of several modules, each of which handles an information source. The design idea is that each module has an information source and related storage space that it handles. This design enables removal and insertion of additional information sources without affecting the other modules.

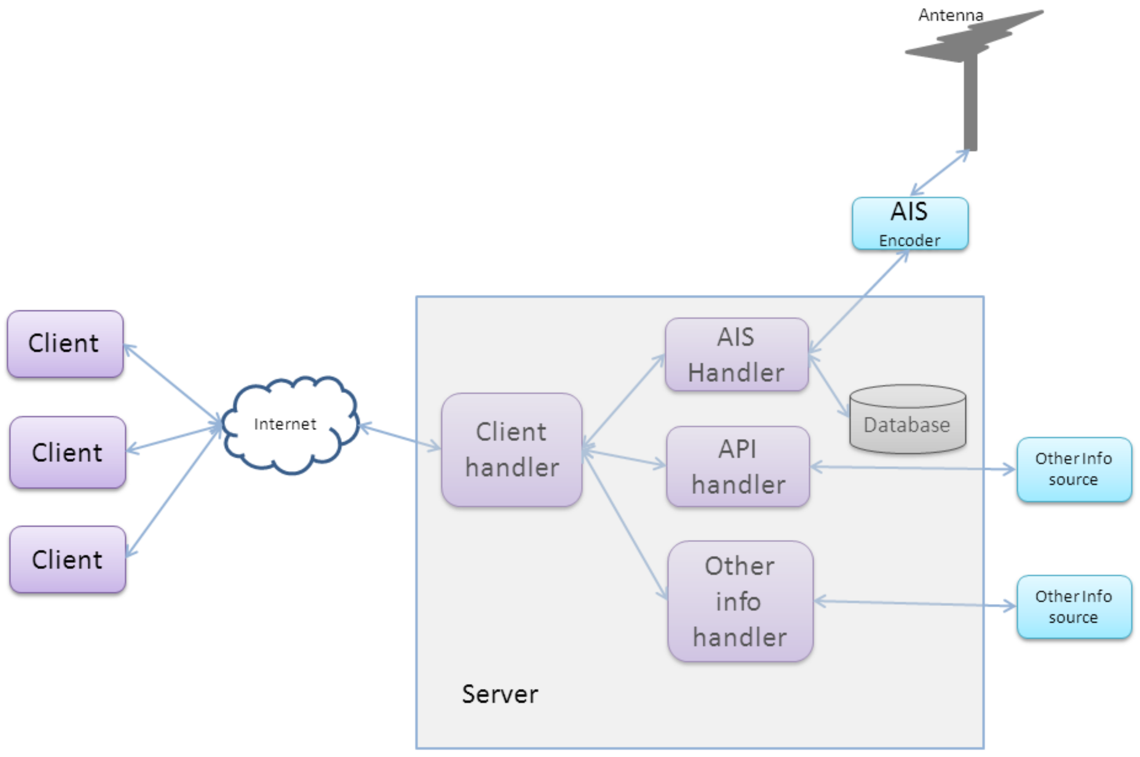


Diagram 1 showing the Domain-level Diagram.

4.5.1 Deployment (physical) Diagram

This diagram focuses on the physical environment that the application is intended to run in. The idea of our model is that a server application runs on a cloud service where it communicates with different information sources as well as clients.

For Ship Detector to operate properly the server application needs to run on a computer device or similar that supports Java. The client application needs to run on a mobile device and in this case it has to be on an Android platform (2.1 or higher).

The server application is placed on a cloud service using a VPS and communicates with the AIS Encoder through internet connection. The same type of communication is held between the client and the server on a different port. Additionally, the server communicates with three different web services, APIs, and this is also conducted through internet connection.

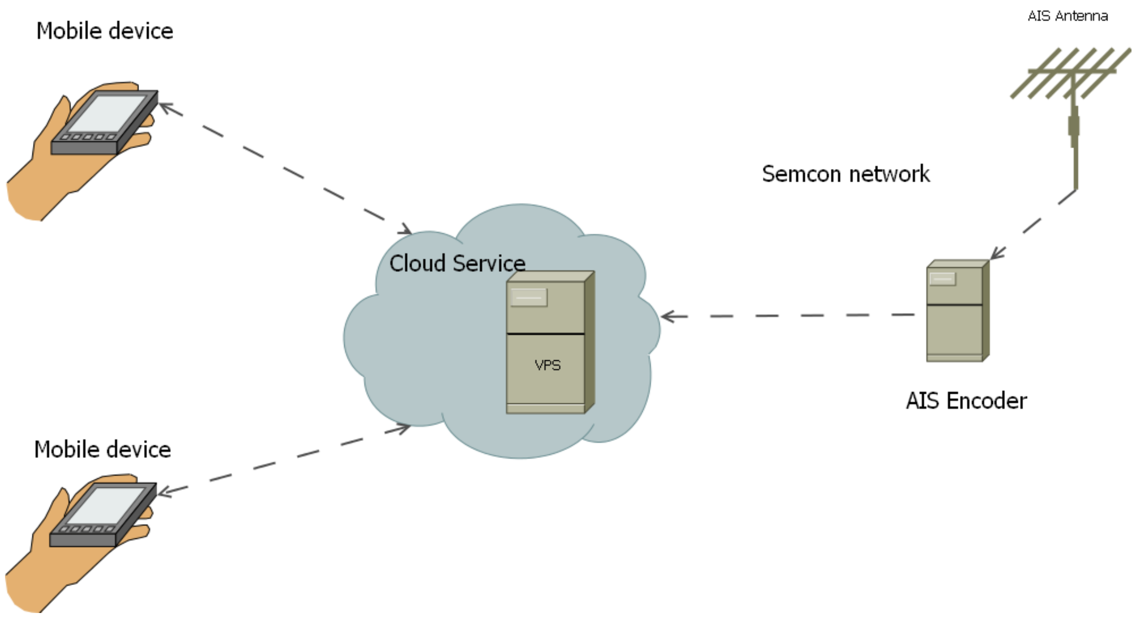


Diagram 2 showing the Deployment Diagram

4.5.2 Functional (logical) Diagram

A functional diagram describes the application's functional elements, their responsibilities, interfaces and primary interactions (Rozanski et al. 2005). It can also be seen as the logical viewpoint of the application. UML (Unified Modelling language) is used to document the functional structure of the application. As shown in the table below Ship Detector is divided into a client (pink components) and a server (grey components). In addition, the external information sources are represented in blue components. The AIS encoder is an external application that receives and encodes AIS signals to send them through to Ship Detector. The arrows represent dependencies.

Component Description

Client

User interface

This component represents the classes that handle GUI related functions. Since this is an Android application, the user interface component contains a number of activity classes and xml files that make up the GUI.

Client Handler

This component contains classes that handle the connection between the GUI the database and the server.

Database Manager

This component contains classes that handle the insertion and extraction of data from the database as well as the connection between the database and the rest of the client application.

SQLite DBMS

This component represents the actual database management system SQLite.

Server

AIS Signal Handler

This component receives AIS signals, converts them to java objects and extracts relevant information before forwarding them to database manager for storage.

API Handler

This component contains classes that handle requests to and responses from web services. It receives responses in JSON format and converts them into java objects before sending them through to the client manager.

Client Manager
The Client Manager component handles the connections with the clients, and also deals with the client enquires. Depending on the type of enquiry, the Client Manager will contact the API Manager or/and the Database Manager.

Database Manager
The database manager is responsible for insertion and extraction of data from the database and also for the logic part of the database system.

Database Derby DBMS
This component represents the actual database management system SQLite.

External Components
The external components include the AIS Encoder and the web services that the server application contacts including Västtrafik, CommuteGreener and Google Maps.

When an end user chooses to view vessel information, the client application retrieves this information from its local database. This database is automatically updated from the central database on the server application at regular intervals.

As depicted in the diagram, the request and response between the client and the server is passed on from one module to another. Thus, a component only communicates with the component that is above and below it.

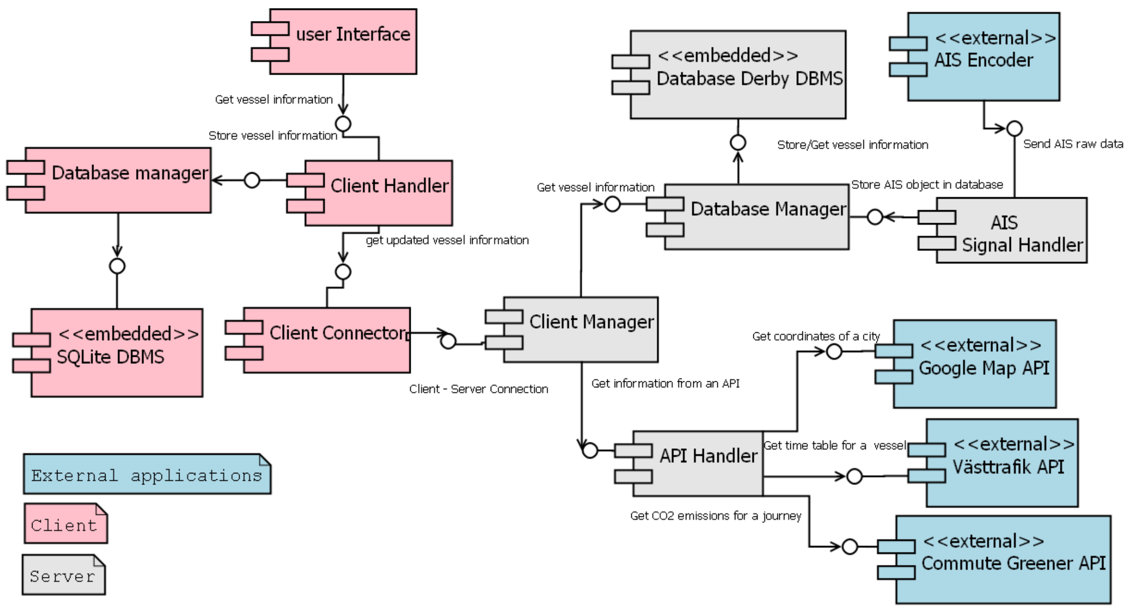
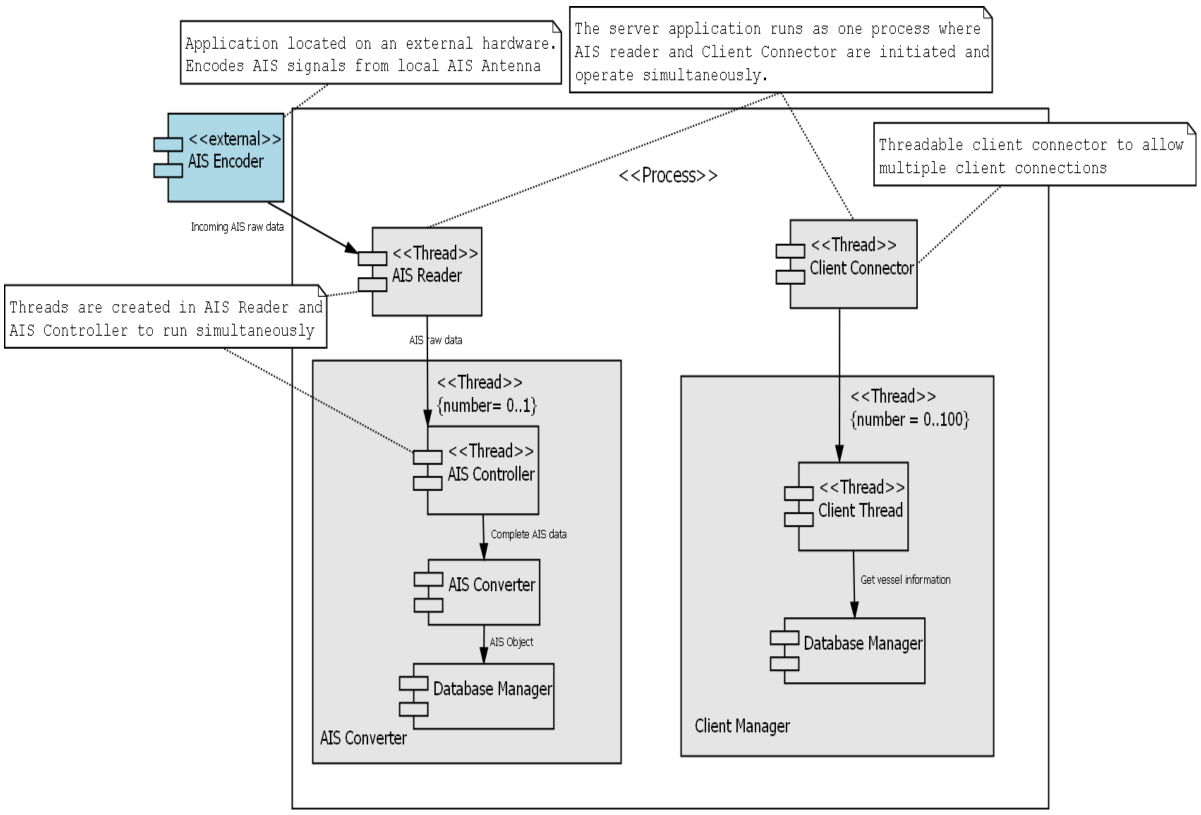

user Interface
Get vessel information
Store vessel information
Database manager
Client Handler
get updated vessel information
<<embedded>> SQLite DBMS
Client Connector
Client - Server Connection
Client Manager
Get information from an API
<<embedded>> Database Derby DBMS
Store/Get vessel information
Get vessel information
Database Manager
Store AIS object in database
<<external>> AIS Encoder
Send AIS raw data
AIS Signal Handler
Get coordinates of a city
<<external>> Google Map API
API Handler
Get time table for a vessel
<<external>> Västtrafik API
Get CO2 emissions for a journey
<<external>> Commute Greener API
External applications
Client
Server

Diagram 3 showing the functional Diagram

4.5.3 Concurrency (dynamic) Diagram

This view describes the concurrency structure of Ship Detector. The application has a few concurrent elements as indicated in the table below (the parts that are stereotyped as “Thread” indicates concurrency). As shown in the table, when the application is initiated, two parts run simultaneously, AIS Reader and Client Connector. AIS Reader, which is the element of the server application that receives AIS raw data, runs as a thread. This in turn creates yet another thread of AIS Controller, which is the part where the system checks if the received AIS message is complete. If this is not the case it will wait for the remaining part of the message to be received before passing it on to the next stage.

Client Connector handles the communication with the client part of the application. It initiates a socket connection on a certain port, and waits for incoming client connection requests. When this occurs, a client thread is created that runs concurrently with Client Connector. This is to allow multiple client connections.
The Java Derby database is threadable and can handle concurrent queries. On the client side, there are no concurrent elements.



Application located on an external hardware. Encodes AIS signals from local AIS Antenna
The server application runs as one process where AIS reader and Client Connector are initiated and operate simultaneously.
Threadable client connector to allow multiple client connections
<<external>> AIS Encoder
<<Process>>
Incoming AIS raw data
<<Thread>> AIS Reader
<<Thread>> Client Connector
Threads are created in AIS Reader and AIS Controller to run simultaneously
AIS raw data
<<Thread>> {number= 0..1}
<<Thread>> AIS Controller
Complete AIS data
AIS Converter
AIS Object
Database Manager
AIS Converter
<<Thread>> {number = 0..100}
<<Thread>> Client Thread
Get vessel information
Database Manager
Client Manager

Diagram 4 showing the Concurrency Diagram

4.5.4 Operational Diagram

The operational viewpoint describes how an application will be operated, administered and supported when it is running on its production environment (Rozanski et al 2005). Ship Detector is written in java, android related java and SQL. The client and the server run as two separate applications and require different libraries and development tools.

For the server application, development needs to take place in a java development tool, with JDK 1.6. A Apache derby dependency needs to be installed and added to the application in order to manage the derby database. A JSON library is needed for converting JSON formatted code to java code. Another dependency is the httpclient, httpcore and commons-logging for making use of the external web services (APIs).

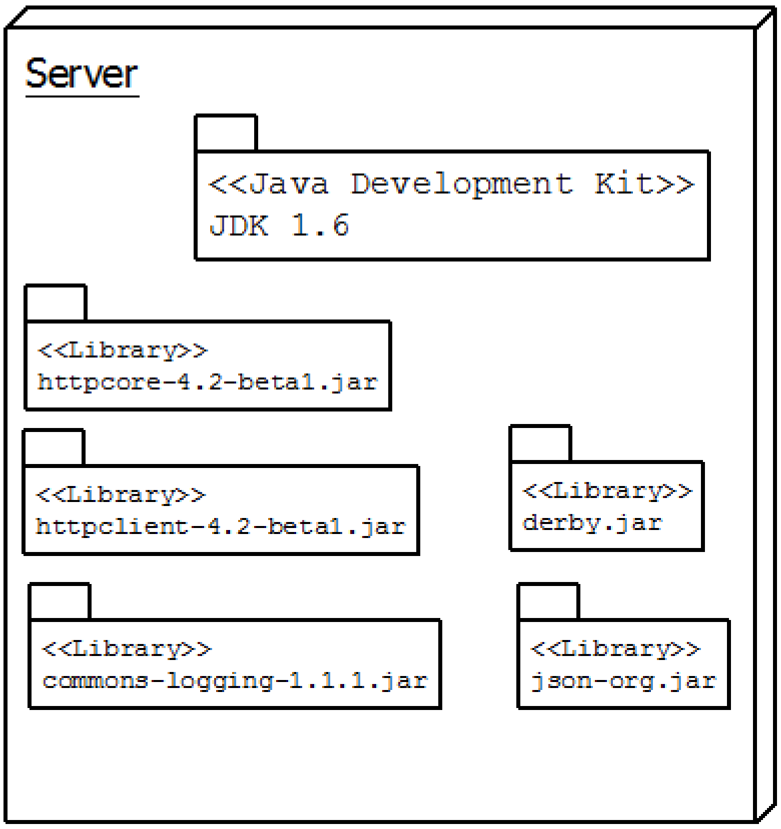


Diagram 5 showing the server Operational Diagram

On the client application, apart from a java development kit JDK 1.6, Android SDK, software development kit needs to be installed with version 2.1. To make Eclipse a more powerful environment for developing Android the ADT (Android Development Tools) plugin was installed. Since Ship Detector makes use of Google maps, this library was added to the application when creating a project that targets Google APIs. With this the keystore was included to provide a key to give permission to use Google maps. A virtual development tool may also be needed for testing the application (unless a real device is not used). This must also be installed and requires Google API version 7. The device on which the production environment is running has to have network connectivity and have available ports to be used.

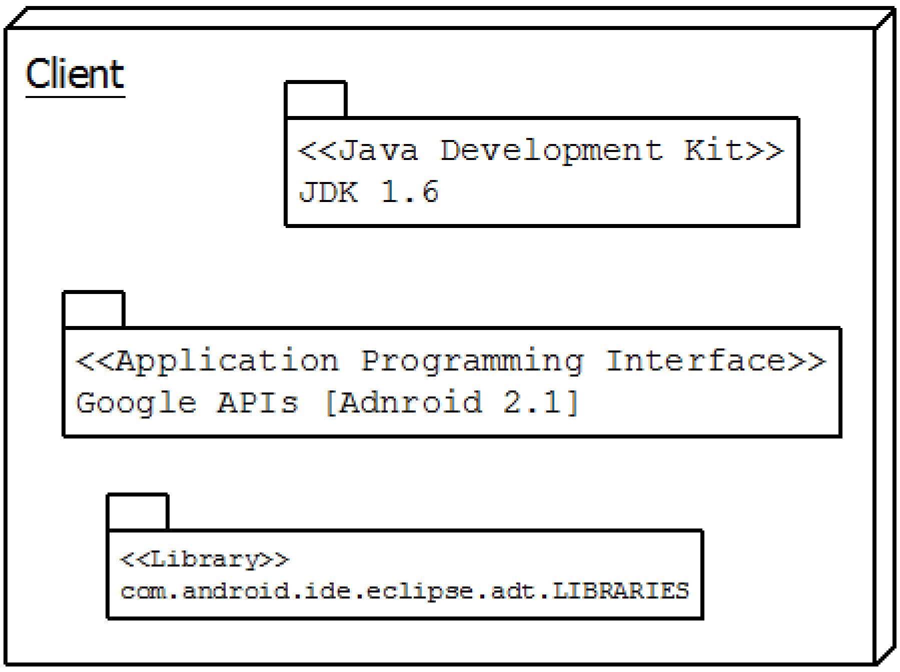

Client
<<Java Development Kit>>
JDK 1.6
<<Application Programming Interface>>
Google APIs [Adnroid 2.1]
<<Library>>
com.android.ide.eclipse.adt.LIBRARIES

Diagram 6 showing the client Operational Diagram

5 Result

The smartphone market is growing rapidly and the range of mobile applications provided is numerous. Meanwhile the market is offering an increasing number of network connected products and it is predicted that in the next decade a large number of appliances will have network connectivity through which they will be monitored and maintained. The challenge is to find a suitable modular way to connect these products to computers and mobile platforms in order to be able to access their information and to monitor/control them directly through these devices.

Accordingly, our aim was to develop a software model for an application that is able to connect a mobile phone to another device and allow the mobile user to access information generated by that device. This software model was to be made modular in order to be used as a concept for similar applications. Furthermore, the software model was going to be documented through proper and understandable diagrams clearly depicting the different technical and design levels of the application.

The study resulted in the development of a working application, Ship Detector, which is an android mobile application that displays information about vessels in and around Gothenburg. This vessel information is taken from AIS signals through an antenna which is localised in the Semcon building in Lindholmen.

As indicated in the domain level diagram above the application adopts the client server model whereby the server, which can be run on a computer device, is the connection point between the AIS Encoder and the mobile device, the client. Furthermore, the server handles other information sources, such as APIs from Västtrafik and CommuteGreener.

In terms of the modular characteristics of our software model, the aim was to be able to reuse this model in order to make other product to computer device connections. As indicated in the domain level diagram above, on the server part different information sources are handled separately to allow insertion of additional information sources and to allow removal of information sources without affecting one another. Thus, every information source handler is a separate component. As shown, the AIS handler is separate from the API handler and so any alterations in either component will not affect the other.

On the client part, we attempted to separate the GUI component from the logic component so that the GUI is only concerned with displaying the information, as indicated in the chapter Application model. However, the client part of Ship Detector is bound to the Android platform as it is written in java and if is going to be used on other platforms then it has to be rewritten in their respective programming languages.

To see a detailed description of the software component a class diagram can be found in appendix 5.

6 Conclusion

As indicated in the report we were able to build a model that could be used as a concept for developing applications that connect electronic products to PCs and mobile platforms, allowing them to be monitored and controlled through these devices. This model was designed in a modular way in which additional products could be easily added, modified and removed without greatly impacting the rest of the model. Moreover, we demonstrated this model through developing an application, Ship Detector, that incorporated these concepts as was described earlier in the report. The design of the application was presented in five different diagrams depicting its different technical levels and perspectives.

Several challenges were faced during the development process of this study and these are discussed in the following section.

6.1 Critical Discussion

Since we already had experience in many aspects of the application that we developed such as Android development and concepts such as server-client communication and design it was rather easy to set suitable time estimations. However we did fail to make realistic time estimates on the parts of the project that we had little experience on. This resulted in cutting out some features towards the end. Some of these features are discussed in the future development section.

The main challenge with this project was to build a good architecture for an application that connects products to PCs and mobile platforms. But how would we know what was a good architectural solution? Which characteristics should our architecture have and which characteristics should we focus on and why? The questions were addressed by consulting software architects and other experienced professionals as well as reading relevant literature. We took into account some of the advice we received as well as following some design decisions that have been previously built and tested and proven to work well.

When constructing a software architectural design there are numerous decisions that must be made. Customarily all these decisions are made before developers start to develop the system. All documentation is created in advance to make it possible for stakeholders to see how the end product will look and enable them to contribute with input. This was a rather difficult task for us as we didn’t have much experience in software architecture. It also made it hard to balance this type of approach with the agile development method.

As discussed earlier in the report, agile software development is highly preferred amongst developers. Agile development allows the team to plan for one period at a time, thus setting out the requirements, functions and priorities specific for that upcoming period. This allows flexibility where clients can change and add requirements to fit their needs during the time of development. However, to some extent, this goes against the notion of good architectural design making. It is regarded that the whole system should be built up before beginning the development process so that all stakeholders can have their influence and input on the software. But a Scrum master will let the architecture emerge during the development process. This is a problem that we had to deal with. How do we create a good architecture if we take it as it comes? We discovered that by making the architecture modular as opposed to integrated, adding new functions and changing priorities did not pose a hinder on the development of the application. We found that agile development required good architecture and so we were able to detect whether our architecture was suitable when the application faced additional requirements and modifications.

As mentioned earlier in this report to demonstrate the architecture we designed a server-client application that required network communication between the computer device running the server application and the Android device running the client application. We realized early on in the process that the android phones that were connected to mobile networks would not be able to connect to the server as it was running inside Semcon’s highly secured network. The IT security in Semcon posed restrictions when testing the client server communication. We were therefore only able to test the application with an android emulator and when using the emulator environment we could not simulate the GPS and other embedded functions so the application could not be completely tested. This testing problem was later solved by setting up the server application on a VPS.

When looking back on the process there are some things that we could have done differently to get a better outcome. The result might not have been different but the development process could have gone smoother had we outlined our requirements on the Semcon network and been given permission from the IT department. With this approach the use of the cloud service would have been up and running sooner and the testing of the application would had been easier and faster and we could have tested all features at the same time instead of making alternative testing.

6.2 Generalization

The goal with this project was to develop an application model for connecting products to computer devices and this model was going to be used as a concept for applications with similar functions. Therefore the generalisation of the model was taken into account at the start of the development process.

It is predicted that more and more appliances will have network connectivity in the next decade. These devices will include appliances such as refrigerators and washing machines. By connecting these appliances to remote computers this can for example allow providers to efficiently monitor and maintain these appliances from a remote site. These appliances can also be connected to mobile devices and become accessible at all time, sending alerts and other valuable information that can be addressed more promptly. This type of mobile device connection will also enable a more mobile working environment. Instead of sitting and monitoring a system, employees can occupy themselves with other tasks or be at other places and still receive system updates and other relevant information through the mobile device.

6.3 Future development

As mentioned earlier, there were some features that were not implemented due to time constraints. For example future work can be done on improving the graphical interface of Ship Detector to make it more user friendly and marketable.

An additional feature that we did not have time to implement was a function that allows a user to point at a vessel with the mobile device and displays its information. This function would require the GPS and compass features of the phone and possibly a camera if wanting to simulate picture recognition.

Another feature that could be implemented is mapping all the destinations of a vessel on Google map by drawing lines between the destinations. There is currently a database table that persists all the registered destinations for each vessel.

Ship detector is specific to the Android mobile platform, and so a useful area to explore is how to make the application applicable to different platforms. Developing the same application for different mobile platforms is time consuming and costly and therefore cross platform development would make mobile applications more modular.

At this moment the Ship Detector server adopts a passive server model, which means that when the server is contacted by a client the server only responds to the request. Hence the server cannot initiate contact with the client. If an application that adopts our model was going to have alert functions or any other features that require the server to contact the client, there needs to be an implementation of an active server.

Moreover, Ship Detector was development to be used as a concept for similar applications. Future development in this area would mean using the principles of this application model and connecting other products that can provide more useful information.

References

Bass, L., P.Clements, R.Kazman. (2007) *Software Architecture in Practice,* 2nd ed. Boston : Addison-Wesley. (SEi series in Software Engineering)

Beedle M., Bennekum A., Cockburn L., Cunningham W., Fowler M., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R. C. , Schwaber K., Sutherland J., Thomas D. (2001). *www.agilemanifesto.org.* http://agilemanifesto.org/history.html (14 Mars 2012)

Ericsson, Networked Society. (2012) Shaping Sustainable Cities in the Networked Society. *http://www.ericsson.com/networkedsociety/.* http://www.ericsson.com/networkedsociety/media/hosting/Sustainable_Networked_Cities.pdf (29 May 2012)

Gobry, P-E. (2012) ANALYST: Smartphone Sales will Dwarf PC Sales this Year and Reach a Staggering 1.5 Billion per Year by 2016. *www.businessinsider.com.* http://articles.businessinsider.com/2012-02-29/tech/31109577_1_smartphones-pc-sales-internet#ixzz1wLBSqWB5. (2 May 2012)

Helal, S., Bose, R., Li, W. (2012) *Mobile platforms and development environments.* [Electronic] Florida: Morgan & Claypool. (Synthesis digital library of engineering and computer science. Synthesis lectures on mobile and pervasive computing ; # 9.)

International Maritime Organization (IMO) (n.d.) International Convention for the Safety of Life at Sea (SOLAS), 1974. *http://www.imo.org.* http://www.imo.org/about/conventions/listofconventions/pages/international-convention-for-the-safety-of-life-at-sea-(solas),-1974.aspx. (30 May 2012)

Kyocera QCP 6035 Smartphone (2001) *www.Palminfocenter.com.* http://www.palminfocenter.com/view_story.asp?ID=1707. (7 May 2012)

Orenstein, D. (2002) QuickStudy: Application Programming Interface (API). *http://www.computerworld.com/*.http://www.computerworld.com/s/article/43487/Application_Programming_Interface?taxonomyId=11&pageNumber=1. (30 May 2012)

Rozanski, N.,Woods, E. (2005*) Software Systems Architecture : working with stakeholders using viewpoints and perspective.* Upper Saddle River, USA:Pearson Education, INC

Schwaber, K., Sutherland, J. (2011) The Scrum Guide, The Definitive Guide to Scrum: The Rules of the Game. *www.scrum.org.* http://www.scrum.org/storage/scrumguides/Scrum_Guide.pdf. (16 April 2012)

Sjöfarstverket (2012) AIS transpondersystem. *http://www.sjofartsverket.se* http://www.sjofartsverket.se/Infrastruktur-amp-Sjotrafik/Sjotrafik information/AIS-transpondersystem/ (12 April 2012)

Smartphone definition from PC Magazine Encyclopedia (n.d.) *http://www.pcmag.com/*. http://www.pcmag.com/encyclopedia_ term/0,2542,t=Smartphone&i=51537,00.asp. (11 May 2012)

Appendix 1 – AIS Table

message	description	sending rate
MMSI	unique id	x
Navigation status	ex "at anchor", "user way using engine(s)"	x
Rate of turn	right or left, from 0 to 720 degrees per minute	x
Speed over ground	0 to 102 knots	x
Position	Longitude and Latitude	x
Course over ground	relative to true north to 0.1°	x
True heading	0 to 359 degrees	x
UTC Seconds	seconds ago data were generated	x
IMO	seven digit number	6 min
Radio call sign	x	6 min
Name	name of the ship	6 min
Type	type of ship/cargo	6 min
Dimension of ship	to the nearest meter	6 min
Location of positioning systems (e.g. GPS)	meters aft of bow and meter port of starboard	6 min
Type of positioning system	such as GPS, DGPS or LORAN-C	6 min
Draught of ship	0.1 meter to 25.5 meters	6 min
Destination	max. 20 characters	6 min
ETA (estimated time of arrival) at destination	UTC month/date hour:minute	6 min

x: Ship at anchor or moored and not moving faster than 3 knots 3 min
Ship at anchor or moored and moving faster than 3 knots 10 s
Ship 0-14 knots 10 s
Ship 0-14 knots and changing course 3 1/3 s
Ship 14-23 knots 6 s
Ship 14-23 knots and changing course 2 s
Ship >23 knots 2 s
Ship >23 knots and changing course 2 s

Information retrieved from http://en.wikipedia.org/wiki/Automatic_Identification_System#Message_types [19 March 2012]

Appendix 2 - Database

Server database:

Ships(<u>MMSI</u>, name, ETA, # length, #width, navStat, destination, #latitude, #longitude, #draught, updated, #rot, shiptype, #cog, #speed)

Description – Ship table
The Ship table consists of all information that is received from the AIS signals, every ship consist of one row in the table. When signals are obtained from a ship that never has been in the database a new row is created, but if a signal is received from a known ship that ships row is updated with the new information. The key is used to be able to get precisely one row and for this the ship's unique identification number (MMSI) is used.

Positions(<u>ship</u>, #<u>posnr</u>, #latitude, #longitude)
Ship --> Ships.MMSI

Description – Position table
In the Position table the ten latest positions from a ship is stored to be able to map the direction the ship came from. To get hold of a specific row two keys must be used because every ship has ten positions. The keys in this table are the unique identification number (MMSI) of the ship and the order number 1-10. When a new position arrives the first number is removed and the new is inserted.

Destinations(<u>ship</u>, <u>ETA</u>, destination)
Ship --> Ships.MMSI

Description – Destination table
The destination table stores all the destinations of each ship. Two keys are used, the unique identification number (MMSI) and estimated time of arrival (ETA). This table is currently not in use.

Appendix 3 - Database

Client database

Description – Ship table
This table is designed in the same way as Ship table on the server. Only difference is that the vessel information is updated at fixed intervals so AIS signals from ships that are not within the range of the antenna will not be in the database.

Description – Position table
Same as the Position table on the server but only with updated ships.

Description – Favourite table
The Favourite table lists the unique identification number of a ship that the user has chosen as one of the favourites. “Favourite” marked vessels will be displayed in another colour.

Appendix 4 - Dependencies

Dependencies

Server
httpcore-4.2-beta1.jar
http://hc.apache.org/downloads.cgi

httpclient-4.2-beta1.jar
http://hc.apache.org/downloads.cgi

commons-logging-1.1.1.jar
http://commons.apache.org/logging/download_logging.cgi

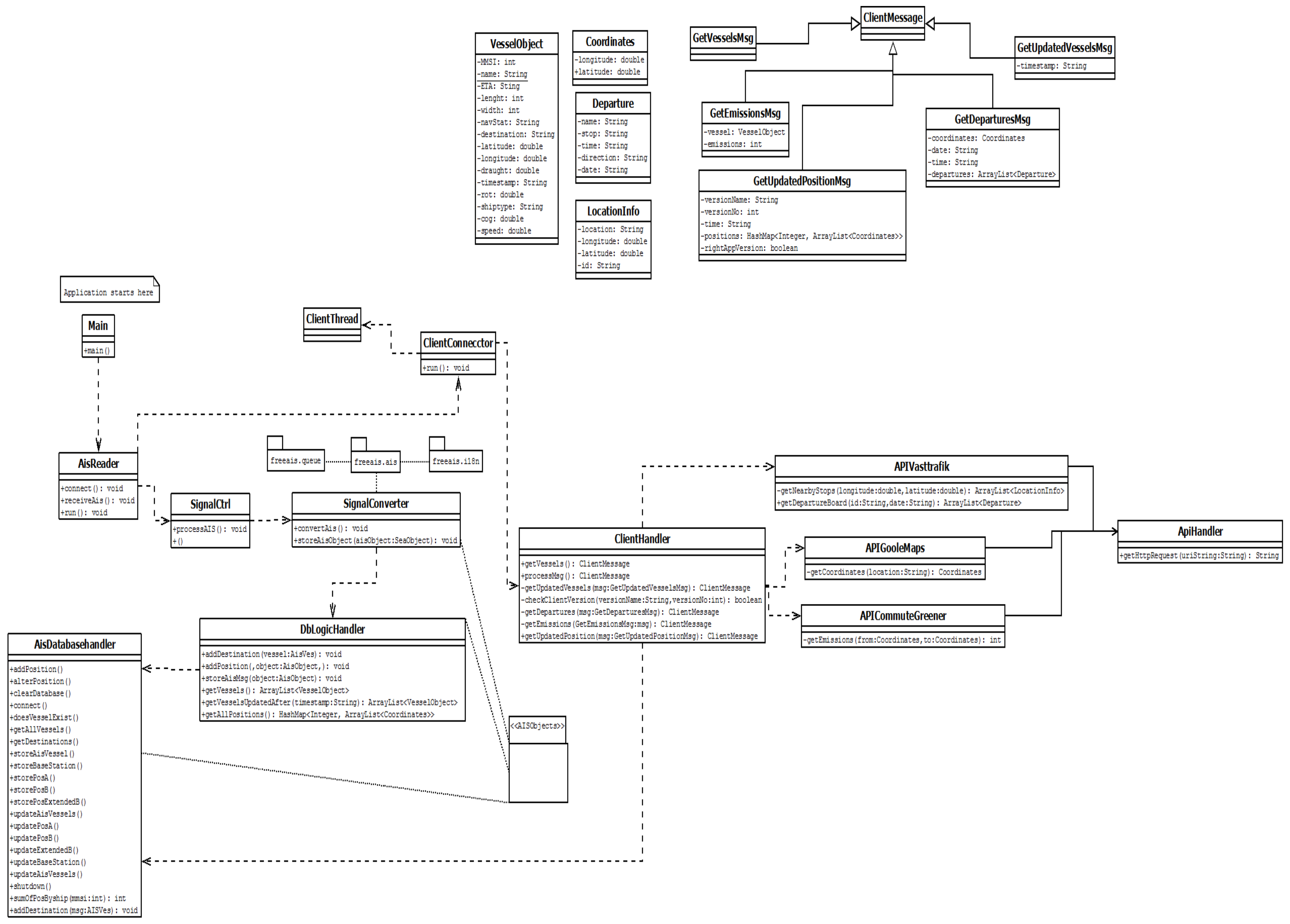
derby.jar
http://db.apache.org/derby/derby_downloads.html

json-org.jar
http://www.docjar.com/jar/json-org.jar

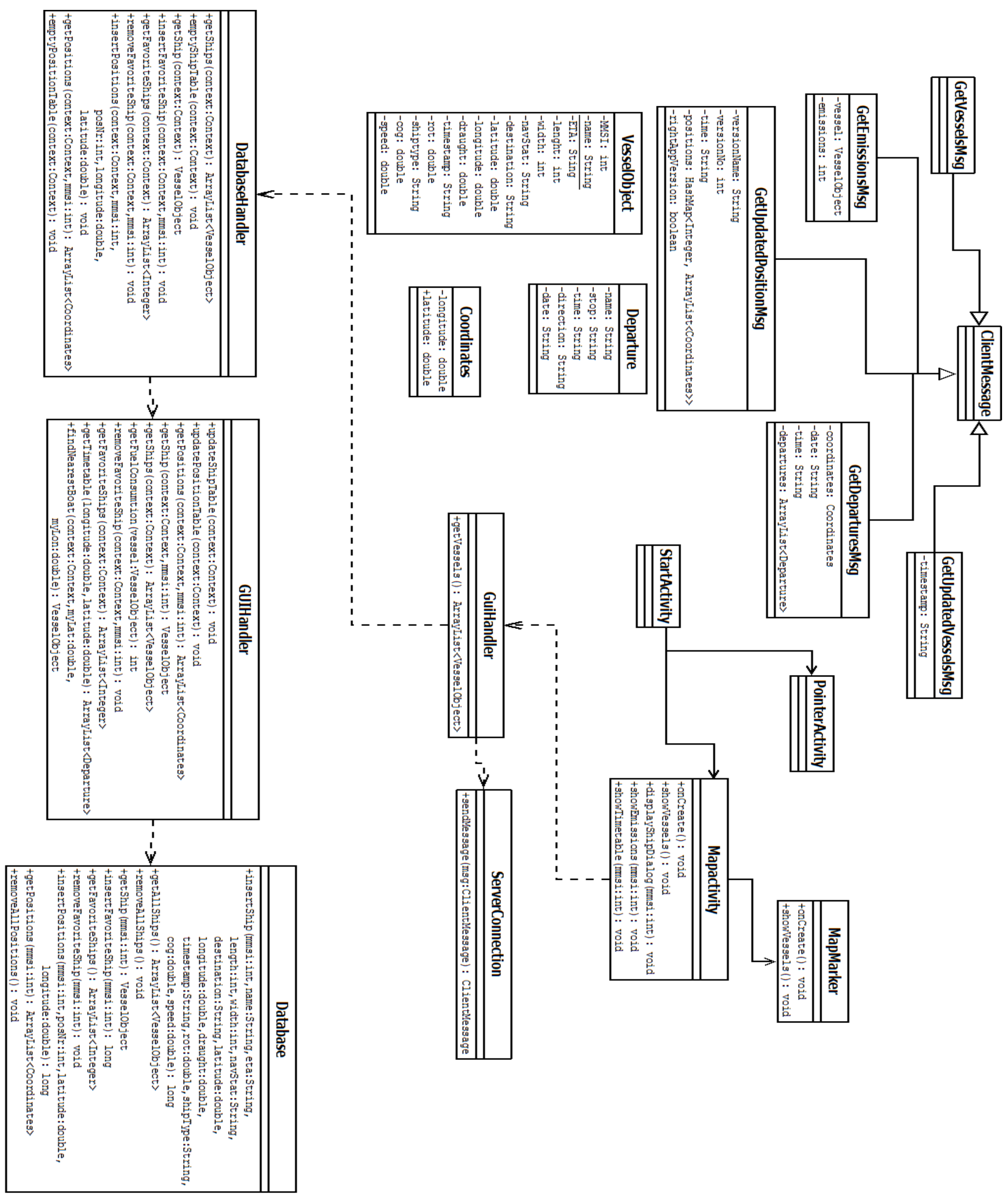
Client
SDK
http://developer.android.com/sdk/index.html

ADT
http://developer.android.com/sdk/eclipse-adt.html#installing

Appendix 5 – Server Class Diagram



Appendix 6 – Client Class Diagram



DatabaseHandler
+getShips(context:Context): ArrayList<VesselObject>
+emptyShipTable(context:Context): void
+getShip(context:Context): VesselObject
+insertFavoriteShip(context:Context,mmsi:int): void
+getFavoriteShips(context:Context): ArrayList<Integer>
+removeFavoriteShip(context:Context,mmsi:int): void
+insertPositions(context:Context,mmsi:int, posNr:int,longitude:double, latitude:double): void
+getPositions(context:Context,mmsi:int): ArrayList<Coordinates>
+emptyPositionTable(context:Context): void
GUIHandler
+updateShipTable(context:Context): void
+updatePositionTable(context:Context): void
+getPositions(context:Context,mmsi:int): ArrayList<Coordinates>
+getShip(context:Context,mmsi:int): VesselObject
+getShips(context:Context): ArrayList<VesselObject>
+getFuelConsumtion(vessel:VesselObject): int
+removeFavoriteShip(context:Context,mmsi:int): void
+getFavoriteShips(context:Context): ArrayList<Integer>
+getTimetable(longitude:double,latitude:double): ArrayList<Departure>
+findNearestBoat(context:Context,myLat:double, myLon:double): VesselObject
Database
+insertShip(mmsi:int,name:String,eta:String, length:int,width:int,navStat:String, destination:String,latitude:double, longitude:double,draught:double, timestamp:String,rot:double,shipType:String, cog:double,speed:double): long
+getAllShips(): ArrayList<VesselObject>
+removeAllShips(): void
+getShip(mmsi:int): VesselObject
+insertFavoriteShip(mmsi:int): long
+getFavoriteShips(): ArrayList<Integer>
+removeFavoriteShip(mmsi:int): void
+insertPositions(mmsi:int,posNr:int,latitude:double, longitude:double): long
+getPositions(mmsi:int): ArrayList<Coordinates>
+removeAllPositions(): void
VesselObject
-MMSI: int
-name: String
-ETA: Sting
-lenght: int
-width: int
-navStat: String
-destination: String
-latitude: double
-longitude: double
-draught: double
-timestamp: String
-rot: double
-shiptype: String
-cog: double
-speed: double
Coordinates
-longitude: double
+latitude: double
Departure
-name: String
-stop: String
-time: String
-direction: String
-date: String
GuiHandler
+getVessels(): ArrayList<VesselObject>
ServerConnection
+sendMessage(msg:ClientMessage): ClientMessage
GetUpdatedPositionMsg
-versionName: String
-versionNo: int
-time: String
-positions: HashMap<Integer, ArrayList<Coordinates>>
-rightAppVersion: boolean
GetEmissionsMsg
-vessel: VesselObject
-emissions: int
GetVesselsMsg
ClientMessage
GetDeparturesMsg
-coordinates: Coordinates
-date: String
-time: String
-departures: ArrayList<Departure>
GetUpdatedVesselsMsg
-timestamp: String
StartActivity
PointerActivity
Mapactivity
+onCreate(): void
+showVessels(): void
+displayShipDialog(mmsi:int): void
+showEmissions(mmsi:int): void
+showTimetable(mmsi:int): void
MapMarker
+onCreate(): void
+showVessels(): void