CHALMERS

Extensible in-vehicle ITS system for location based traffic information

A design proposal and implementation for the NS-FRITS project

Master of Science Thesis in Networks and Distributed Systems

Jonas Färdig
Carl Jonsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2010

Extensible in-vehicle ITS system for location based traffic information.
A design proposal and implementation for the NS-FRITS project.

Jonas Färdig
Carl Jonsson

Examiner: Arne Dahlberg

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

Department of Computer Science and Engineering
Göteborg, Sweden June 2010

Abstract

As the European population grows, the freight movement and transport sector in the North Sea Region increases, leading to a higher load on existing road networks, ports and intermodal interchanges. Rather than only focusing on expanding existing physical infrastructure, the introduction of Intelligent Transport Systems (ITS) have been predicted to have crucial role in improving efficiency and safety in the transport sector.

In 2009 the European Union started the NS-FRITS project, which aims to develop an ITS system to enable drivers and fleet managers in the North Sea Region to get location based information about current events such as road conditions, road weather and traffic congestions. The NS-FRITS system should also present information about secure parking spots, crime spots, customs stations and other Point of interests relevant to heavy goods drivers.

The work of this thesis has suggested a system design and architecture of the NS-FRITS system and implemented it into a working prototype. The implementation includes: A server providing location based services, a graphical interface for adding location based information, a client application to be used in trucks and a truck fleet management system. The communication is based on streaming XML through the Extensible Messaging and Presence Protocol (XMPP) protocol.

This report will describe this proposed design, what technologies and protocols are being used, alternative technologies, and suggestion for improvements towards the final NS-FRITS system, which is scheduled to be completed in December 2011.

This report is written in English.

Preface

This report emerges from a master thesis work under the department of Computer Science and Engineering (CSE) at Chalmers University of Technology, and was conducted at Volvo Technology, both located in Göteborg, Sweden.

We would like to thank our examiner Arne Dahlberg at the department of computer science and engineering, Chalmers University of Technology and the people at Tullverket in Svinesund.

Special thanks goes to our supervisor at Volvo Technology, Claes Pihl.

Contents

1. Introduction

1.1. Background

During the last 20 years, many industries have been transformed by the use of Information Technology (IT). Lately, the transport sector have realized that the key to efficient transport systems lies in the use of IT rather than only focusing on reparation of old infrastructure and building new roads[17].

Intelligent Transport Solutions (ITS) is the way of improving the transport sector by introducing information and communication systems. This is a broad concept that can be realized in many different ways:

- Optimizing vehicle routes based on current road congestions can reduce fuel consumption and transportation time.
- By maximizing the load of each vehicle in a freight company depending on route, will reduce the amount of trips per goods.
- By giving the driver information about Point of Interests (POI), such as safe parking spots or resting places, reduces the down time having to search for them manually. It can also affect planning and driving decisions by the driver and its operator.[28].

Research of ITS systems have been in major focus of many countries during the last years, South Korea spends 100-200 million USD per year in ITS research and China has already deployed ITS systems in major cites such as Beijing, Guangzhou and Shenzhen[23].

In December 2008 the European Union (EU) adopted an action plan containing measures and directives for ITS development. According to the EU, traffic congestions which are estimated to 1% of the European GDP could be reduced with 10% with the introduction of ITS. The EU also argues that ITS systems leads to lower carbon dioxide emissions and could prevent more than 5000 deaths in road accidents[5] by reduced congestion.

1.2. The NS-FRITS Project

In January 2009 the European Union started the NS-FRITS project. NS-FRITS, which stands for North Sea Freight Intelligent Transport Solutions, is a project which aims to improve the accessibility and efficiency for the freight sector in the North Sea region by introducing ITS[37]. The project is a public/private sector partnership with actors

from the UK, Germany, the Netherlands and Sweden. These actors together represents high-tech communication, logistics, crime reduction, education, policing and transport planning.

The goal of the NS-FRITS project is to provide a multilingual system for drivers, transport managers and freight managers. A driver should be able to use an in-vehicle computer or a smartphone connected to the NS-FRITS system while the transport and freight managers are connected to the NS-FRITS system via a desktop computer. The system should be able to provide driver-to-manager communication as well as providing position based information to clients. Amongst the information NS-FRITS will provide include: Poor weather conditions, road accidents, traffic disruptions and general road information.

The NS-FRITS system will also reduce the security threats for the driver by for example providing information about crime hot spots, secure parking locations and local policing practices. The driver should also be able to report security or safety related issues through the NS-FRITS system, such as suspected crime or accidents.

From the freight managers point of view NS-FRITS provides a possibility to contact the driver about driving jobs, get updates about the drivers position and live text communication.

The requirements of the NS-FRITS system has been concluded by a number of interviews with drivers, transport managers and haulers.

1.3. Related Actors

The NS-FRITS project is divided into several work packages, each responsible of a certain area. The main work package that this project has been working with is the package responsible for design and implementation of the NS-FRITS project. This section will describe all actors that this project have cooperated with during the design of the NS-FRITS system.

1.3.1. Volvo Technology

Volvo Technology (VTEC) is the center for innovation, research and development in the Volvo Group[49]. It is located at Lundbystrand and Chalmers Science Park in Göteborg, Sweden and at Volvo's establishments in Lyon, France and Greensboro, USA. Volvo Technology is one of the partners in the NS-FRITS project.

1.3.2. Tullverket - Swedish Customs

Along with the development of this project, contact with the Swedish customs in Svinesund have been made. The Swedish customs is a stakeholder and a potential data

provider for the final NS-FRITS system.

1.4. Purpose

The NS-FRITS project have several purposes, but the main purposes can be simplified to the following: Improving the quality of life for individual workers in the logistics and transport sector and increasing efficiency and lowering environmental impact of the transport sector around the north sea area[37].

The purpose of this thesis is to develop and design a prototype system for the NS-FRITS project. The results of this thesis can then be used as a foundation for further development of the NS-FRITS system or other ITS-projects in the same area. The final NS-FRITS system is scheduled to be completed in the end of 2011.

1.5. Goals

There are two main goals for this thesis:

The first goal is to design a system for collecting information from different data providers and compile it into a format that can be sent to vehicles interested in this information. The design involves specifying an API and a data model for the communication between the vehicles and the system backend.

The second part is to implement parts of this system into a working server backend and a desktop client. The main purpose of this implementation is to be used as a showcase in one of the NS-FRITS pilot demonstrations in June 2010. This implementation could later on be used as a model for future work in the NS-FRITS project.

1.6. Constraints

- The prototype implementation only needs to work on a desktop system, there is no need to implement an actual embedded in-vehicle system.
- All parts of the system will have a constant Internet connection.
- No security implications will be considered since its only a showcase.

1.7. Disposition

The main parts of this report are divided into a problem analysis, a system design, a prototype implementation and an improvements chapter.

The problem analysis chapter presents the requirements of the system and gives a technical background to the technologies and protocols used in the designed system. This chapter also describe alternative technologies and their respective pros and cons.

In the system design chapter, the resulting design of the thesis's work is presented. This chapter explains which technologies were chosen in the final design and argues why. All actors of the NS-FRITS system and their respective roles are described here. In this chapter, the report also goes into details about protocols and data models used.

The prototype implementation chapter explains the implementation of the system, what each application developed in the project do and how it was implemented (programming languages, libaries etc).

The improvements chapter discusses what improvements can be made into the designed system and how it could be extended. Since the NS-FRITS project continues after this thesis work is done, this is considered a large part of the project and is separated from the discussion chapter.

Finally, the results and discussion chapter summarize the work done in this thesis and discusses its outcome.

1.8. Glossary

API
: Application Programming Interface.

CORBA
: Common Object Request Broker Architecture.

DAO
: Data Access Object.

DATEX II
: A European standard specification for traffic data exchange.

GPRS
: General Packet Radio Service, a system for sending data packets over GSM or 3G networks.

GUI
: Graphical User Interface.

ITS
: Intelligent Transport System.

JSON
: JavaScript Object Notation.

NS-FRITS
: North Sea Freight Intelligent Transport Solutions, an EU project in the North Sea area.

OSM
: OpenStreetMaps, a community based map service.

POI
: Point of interest.

Qt
: A cross-platform C++ application and UI framework developed by Nokia.

WKB
: Well Known Binary, the binary form of WKT.

WKT
: Well Known Text, a format for geographical data.

XML
: Extensible Markup Language.

XMPP
: Extensible Messaging and Presence Protocol, a protocol for streaming XML data.

2. Methodology

Since the requirements and goals of this project were not fixed, the development was carried out according to an iterative model rather than a fixed workflow like the waterfall model. The methodologies used in the project can be divided into five different phases: Information gathering, requirements analysis, system design, implementation and evaluation.

The information gathering phase consists of reading existing literature, thesises and documents on the subject and conducting inteviews. In this phase, the project team also researched about data providers, data formats and communication protocols.

In the requirements analysis phase, the project team summarized the requirements of the system and how it would work. Parts of this phase is performed concurrently with the end of the information gathering phase.

During the design phase, the system was designed according to the requirements previously specified. The project team also selected what technologies to use in the implementation phase, and what to use them for.

In the implementation phase, parts of the design were to be implemented into a prototype model of the system. During the implementation stage, the technologies selected in the design stage were being tested and in case of anything not working properly, the design was revisited.

Finally, the implementation of the system were evaluated and tested until it satisfied all requirements of the system. When something did not work as expected, previous stages of the development were revisited until the evaluation succeeded.

3. Problem Analysis

This chapter's goal is to analyze the requirements of the system and present different technologies and techniques relevant to the project. Many different subjects will be discussed, and their respective pros and cons investigated. The results of this chapter was used as a base for the design of the system.

3.1. NS-FRITS Actors and Concept Overview

To get an overview of the concept of NS-FRITS, the main actors of the system will be presented. Three groups of actors, as seen in Figure 3.1, has been recognized in the NS-FRITS system: Data providers, transport managers and drivers.

Data providers are responsible for entering data into the NS-FRITS system, they might include road administration authorities, police stations, weather stations, etc.

Transport managers are units communicating directly with the drivers. A transport manager can see where drivers are located, send jobs to drivers, plan routes and query the NS-FRITS system for information regarding routes. The transport manager role is expected to be handled by hauling or logistic companies, not by NS-FRITS employees.

The driver is the person inside the truck, connecting to the NS-FRITS system with either an onboard computer or a smartphone. Each driver can both query the NS-FRITS system for certain information and receive location based information while driving. Drivers also uses the NS-FRITS system for communicating with their respective transport manager.

3.2. System Requirements

A number of system requirements have been concluded from NS-FRITS project requirements[35][36] and scenarios[34], and from the case study of the Swedish customs office in Svinesund, see Appendix B for more information.

For simulating features and functionality for the client in the NS-FRITS system, a desktop simulator will be developed. By developing it as a desktop application the development time can be reduced since it does not need to be customized for a mobile or otherwise limited hardware platform.

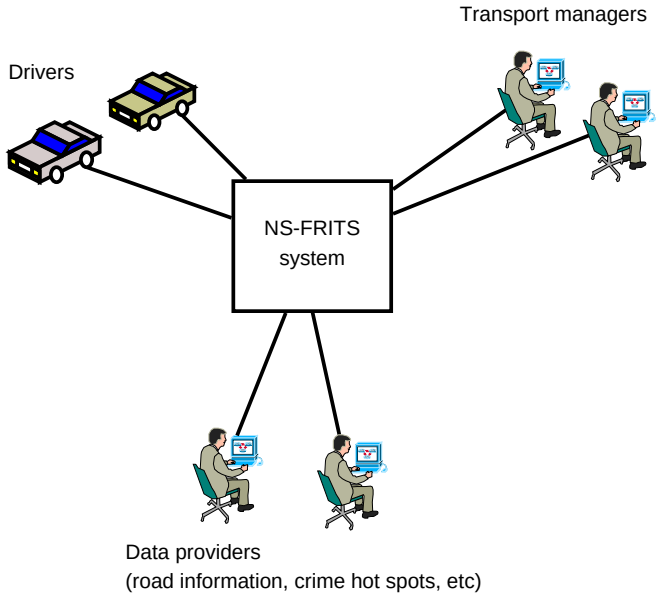

Figure 3.1.: An overview of the NS-FRITS concept and its actors.

A server application will handle the communication to the clients and also import and process the incoming data from the different data providers. A web based administration interface will also be implemented to allow manual editing of the information in the system, this interface will be used by administrators and data providers.

To handle the communication, a simple and extensible protocol will be designed. It should be fairly simple to make the implementation as simple as possible, but still allow for more complex operations by allowing for easy extensibility.

To show the extensibility of the protocol and make the system more interesting in demonstrations a limited fleet management system will be implemented. It will not be described in detail here since it is not part of the core protocol and will only be used for demonstration purposes.

One of the key requirements of the NS-FRITS system is scalability. Since NS-FRITS is expected to work with thousands of simultaneous clients, is it important that technologies and protocols used scale well.

Each requirement has a priority of either mandatory (M) or desirable (D), and is either a functional (F) or non-functional (NF) requirement.

Part	Name	Priority	Category
Client	Set preferred languages	M	F
	Set categories	M	F
	Get available categories	M	F
	Set a route	M	F
	Plan a route	D	F
	Request a route	D	F
	Search for a POI	M	F
	Get updates during trip	M	F
	Play voice messages	D	F
	Display images	D	F
	Display webpages	D	F
	Show information points	M	F
	Send alerts	D	F
	Receive alerts	D	F
	Send current location	D	F
	Receive an order	D	F
	Send order updates	D	F
Server	Receive data from external sources	D	F
	Parse DATEX II data	M	F
	Edit information manually	M	F
	Translate text	M	F
	Store information points	M	F
Fleet management	Show location of trucks	D	F
	Assign orders to trucks	D	F
	Attach routes to orders	D	F
	Show status of trucks	D	F
Common	Support multiple languages	M	NF
	Low bandwidth usage	D	NF
	Scalable server design	D	NF
	Extensible protocol	D	NF

3.3. System Architectures

In the design process of a system there are several different system architectures available for use. A couple of commonly used architectures has been selected for an overview. By analyzing existent architectures and their respective advantages and disadvantages, a suitable architecture can be chosen for the NS-FRITS project prototype.

3.3.1. Client-server Architecture

The Client-server architecture, is the simplest and most commonly used architecture in system design. In this model processes are divided into two different groups, clients and servers[48]. The client sends a request to the server and then waits for a response from the server. When the server receives a request it processes the request and send a reply to the client. A server never initiates the communication with the client, it only sends replies to requesting clients.

Multi-tiered Architectures

One issue with the standard client server architecture is the problem of separating between clients and servers roles. A common scenario is a client sending a request to a server where the server has to gather data from a database to be able to process the clients request. Since the server then has to query the database for the information, this in fact makes the server a client to the database. In the multi-tiered architecture, sometimes called n-tier architecture, the whole process is separated into several logical layers. The most common case of a multitiered architecture involves three different layers, also called Three-tier architecture. Like the name suggest it uses three different layers, presentation layer, application layer and data layer.

The presentation layer is the layer closest to the end user, responsible of presenting the information to the user in an understandable way, for example with a GUI. The second layer, the application layer, is responsible for the application logic and handling user requests. It also moves information between the data layer and the presentation layer. The data layer handles connections with data sources, such as databases or file systems. It is connected with the application layer, either serving it with data after a read, or writing data to the data source.

3.3.2. Distributed architecture

Unlike the client server architecture the distributed architecture, or peer-to-peer architecture, does not have certain nodes dedicated as servers and clients. Instead nodes in the system acts as both clients and servers. The most commonly used method to achieve this is by distributing all processes using a distributed hash table (DHT). In a DHT-system data is mapped to large number keys which are stored among the nodes in the DHT. Any node can retrieve the data by searching the DHT with the key. How this search is performed is implementation dependent. The most notable advantages of using a distributed architecture is that there is no single point of failure and that the system provides high scalability[13] due to the distribution of services. A disadvantage is that distributed solutions often have a higher complexity which can lead to more software errors.

3.3.3. Hybrid architecture

The hybrid architecture is like the name implies a hybrid of the client server architecture and the distributed architecture. Hybrid architectures have some parts of the system distributed, and other parts as client server[48]. One notable system using the hybrid architecture is the file sharing system BitTorrent. In BitTorrent files are split into small pieces, distributed over the network using a DHT. For a node to be able to see which other active node have the requested file, a central server is used called a tracker. The file sharing itself is done in a peer-to-peer manner, while a client server architecture is used between the tracker and the clients.

3.3.4. Stateful vs. stateless server

A server that remembers information about the state that the clients or sessions are in is said to be a *stateful* server[48]. This makes the server more complex since it has to store the states in some way, which could be a problem if it is dealing with a large amount of clients. By storing the state on the server the communication between server and client can be made simpler. A *stateless server* on the other hand, does not store any client states. Each request to a stateless server need to contain enough information in the request itself to get the desired response. An example of a stateless server is a basic web server.

3.3.5. Analysis

Several requirements need to be considered while choosing a suitable architecture for the NS-FRITS system. Some parts of the system is expected to work centralized, such as planning and getting information for a particular route, while other parts, such as communication between the driver and the truck manager can be handled peer-to-peer. The NS-FRITS system should be available with different functionality to several different platforms, so some kind of multi tiered architecture would make it easy to extend. For example, the driver's NS-FRITS client do not have to be very advanced, while the planner/manager need more functionality. The requirement that the driver should be able to receive alerts about local events indicates that at least some parts of the NS-FRITS system need to be stateful.

3.4. Communication Technologies

There are a variety of different communication technologies available for use, this section will discuss a selected part of them and their respective weak and strong areas of use. Since the resulting design will be implemented as an in-vehicle unit communicating outwards, the focus will be on wireless technologies. One of the constraints of the project

was that the prototype only would run on a desktop system, so this section is just meant as a background to existing technologies and does not affect the actual implementation of the prototype. However, the different pros and cons of each technology are considered in the system design phase when different protocols and techniques are chosen.

3.4.1. Mobile telecommunication

GSM

Global System for Mobile Communication (GSM) is the leading standard for mobile communication today, representing over 80% of todays wireless market[51]. GSM is a circuit switched technology but unlike its predecessor it is handles signals digitally, it is therefore considered a second generation (2G) technology. One of the great advantages of GSM networks is the roaming capabilities between networks[47]. Every device has a Subscriber Identity Module (SIM), this allows devices to be bound to a certain subscription. In this way a unit can roam all over the world with one subscription if the mobile unit used supports all frequency bands used in GSM.

GPRS/EDGE Generation 2.5

To be able to handle packet switched data an extension was introduced to GSM, General Packet Radio Service (GPRS). Since GPRS is just an extension to GSM it is often called 2.5G[47]. The transfer rates can reach up to 115 kbit/s[47], but usually around 40-60 kbit/s is to be expected.

Later on an improvement of GPRS called Enhanced Data Rates for GSM Evolution (EDGE) was released. EDGE is deployed as an addon to existing GSM/GPRS networks with data rates up to 384 kbit/s.

3G

Both EDGE and GPRS uses the same frequency spectrum as GSM and is therefore still considered to be part of the 2nd generation of cellular networks[33]. The third generation of systems consist of several different technologies conforming to a standard by the International Telecommunication Union (ITU) called IMT-2000. One of the main groups working to specify third generation systems conforming to the IMT-2000 standard is 3rd Generation Partnership Project (3GPP). One of the earliest 3G technologies is Universal Mobile Telecommunications System (UMTS), where a user can expect data rates from 144-384 kbit/s or up to 2Mbit/s in good conditions[33]. Later on UMTS was extended with High Speed Packet Access (HSPA), providing downlink speed up to 7.2 Mbit/s and uplink up to 5.4 Mbit/s.

LTE and 4G

The next step of mobile telecommunication was Long Term Evolution (LTE) specified by 3GPP[32]. As of April 2010, this is not in wide deployment yet, but several operators have plans for it [3]. LTE downlink speed is specified to reach over 100 Mbit/s and over 50 Mbit/s uplink. The LTE standard also specifies the use Multiple Input Multiple Output (MIMO), which is a technique to improve throughput by using multiple antennas at both senders and receivers side. With MIMO the downlink peak rate is further increased up to 326.4 Mbit/s.

However, LTE does not conform to the 4G standard as specified by ITU. A new standard called LTE-Advanced is being worked on by the 3GPP to conform to these standards. This standard is supposed to be compatible with LTE equipment and its release is expected in 2011. Peak data rates of LTE Advanced is up to 1 Gbit/s downlink and 500 MBit/s downlink[39].

Summary

The main pros of using mobile telecommunication as communication technology is the long range, wide deployment and good roaming capabilities. These properties are especially important for an in-vehicle system. The downsides are the uncertain data rates, the transfer rate of second generation networks are low, but later networks like 3G can reach high data rates. Future networks like LTE and 4G are expected to have even higher data transfer rate. Another downside that often has been discussed most notably in the EU is the high roaming costs for data services between countries. Standard prices goes from €5-10 per megabyte, sometimes even higher[15].

3.4.2. WLAN/IEEE 802.11

A Wireless Local Area Network (WLAN) is a way for several computers to communicate wirelessly with each other. The standard for WLAN:s are specified in IEEE 802.11 by the Institute of Electrical and Electronics Engineers (IEEE). Commonly this standard is referred to just as Wi-Fi (Wireless Fidelity). The original 802.11 standard specified data rates of 1-2 Mbit/s at the 2.4 GHz ISM band[21], but since then new standards has evolved. The 802.11a standard can reach data rates of 54 Mbit/s at 5.7 GHz. The most common standards as of 2010, 802.11b and 802.11g, still operates in the 2.4 GHz band with data rates of 11 respectively 54 Mbit/s.

WLAN:s can work in either ad-hoc or Access point (AP) mode. In ad-hoc mode computers communicate with each other without any intermediate part. The more common mode is the AP mode, here nodes send their data via a special node called Access point. The access point then forwards data to other nodes or the internet. An access point

operate in a certain channel, which is a part of the frequency band used. The 2.4 GHz band allows for 3 non-overlapping channels, with 32 users per access point[21].

The latest standard as of 2010 is the 802.11n standard with peak data rates at 600 Mbit/s and a throughput of over 100 Mbit/s [40]. The higher data rates are achieved with a combination of larger frequency bands and the use of MIMO antennas, the same concept as in LTE.

3.4.3. WAVE/IEEE 802.11p

IEEE 802.11p, also known as Wireless Access in Vehicular Environments (WAVE), is a protocol providing wireless access for moving vehicles[38]. WAVE was originally designed to provide support for ITS applications for the 802.11 standard[29]. These applications involve vehicle-to-vehicle communication, such as alerting vehicle behind of fast breaking, or communication between vehicles and road side units, a road-side-unit can for example alert nearby vehicles of queues or accidents ahead. WAVE uses the Dedicated Short Range Communications (DSRC) spectrum at 5.9 GHz for communication.

3.4.4. CALM

Communications access for land mobiles (CALM) is a ISO approved framework for providing transparent communication over a variety of different technologies like WLAN(802.11a/b/g/n), WAVE(802.11p) and mobile telecommunications(2G/3G/4G)[16]. Like WAVE, CALM's main focus is to provide communication in the ITS sector[27]. The main concept of CALM is to not limit individual stations to a single communication technology, instead a CALM system provides the ability to handover between different technologies.

3.4.5. GPS

The final technology that will be discussed in this section is the Global Positioning System (GPS). GPS is a satellite based positioning system developed by the U.S Department of Defense in the beginning of the 1970:s[14]. Originally GPS was only intended for military use, but later on it was also made available for civilian use. Nowadays GPS functions as a dual system both for military and civilian use. GPS is a passive system, meaning that there only is communication from one side. Because of this GPS can serve an unlimited amount of clients.

There are 24 operational satellites orbiting the earth, each of these sending out messages about their current position and time. Anyone with a GPS receiver and antenna can receive and decode the civilian GPS signals, free of charge. By using trilateration, receivers can calculate their position by receiving the position and time from four different satellites. In theory only three satellites are needed, but in practice four is needed to compensate for the receiver clock offset.

There are many other factors to consider when calculating a position, doppler effect, multipath errors, ionospheric and tropospheric delay, satellite clock and relativistic effects[30]. Several subsystems has been developed to aid the calculation further, like using known fixed ground positions such as cell towers. Accuracy of a standard consumer grade GPS using the civilian system is about 7m in good conditions[30].

3.4.6. Summary

Due to the requirements of being able to reach a driving truck wherever it is, mobile telecommunication is the most probable candidate for NS-FRITS due to its wide coverage. It is also possible to envision the use of WLAN hot spots for larger data downloads, like preloading data for a longer route, although the main parts of the communication will go through mobile telecommunications. WAVE/802.11p/CALM is not as probable option as short range communication, like vehicle-to-vehicle communication, is not a part of the NS-FRITS requirements. However, there could be a use of these techniques, by for example sending information about road conditions or accidents through road side units. GPS will almost certainly be used by the drivers NS-FRITS client, because it needs to know where the user is located so the correct information can be displayed. The operator or fleet manager however, do not need any GPS unit for NS-FRITS to function normally.

3.5. Map projection

To be able to provide location based services there is a need for a system that handle maps and coordinates.

3.5.1. Map providers

There are several different ways of implementing map support in an application. Commercial map databases such as Navteq and Tele Atlas are commonly used by navigational GPS providers such as TomTom and Garmin and proprietary web map services such as Google Maps and Bing Maps. Amongst the free alternative there is OpenStreetMaps[1], which is a collaborative project that aims to create a free road map spanning the entire world. Map databases services generally works by symbolizing areas and roads as polygons and vectors. The maps are further divided into square sized tiles which are stored in a database to avoid too much compilation by the client.

[1] `http://www.openstreetmap.org/`

Listing 3.1: Example of WKT data.

```
POLYGON((11.33514404312 58.845779573283,
11.33514404312 59.162607462219,
11.142883300941 59.162607462219,
11.142883300941 58.845779573283,
11.33514404312 58.845779573283))
```

3.5.2. Geographical coordinate standards

Modeling a spheroid like the earth is not a trivial task, many different methods exists for this purpose. GPS uses a reference system called WGS84, which models the earth as an spheroid with a slightly higher radius at the equator. This system uses standard latitude and longitude coordinates with the zero degree longitude meridian around 100m from the Greenwich meridian. The WGS84 system is also referenced by its European Petroleum Survey Group (EPSG) code "EPSG:4326". [12]

The other broadly used coordinate system is the Mercator system, or with its EPSG code "EPSG:900913". This system is used by several web map services such as Google Maps, Bing Maps and OpenStreetMaps. Unlike the WGS84 projection the Mercator system treats the earth as a sphere and uses x/y coordinates instead of latitude/longitude values. Mercator uses a cylindrical approach to project the map. This can be illustrated by placing the earth inside a cylinder perpendicular to the earths meridians, then drawing the viewable area onto the cylinder. In this projection areas closer to the poles will be distorted, appearing larger than they actually are.

3.5.3. Geographical data formats

The simplest way of modeling geographical features is by just storing its coordinates together with data or an identifier. However this type of approach can be very limiting when modeling more complex structures. For this purpose several specialized data formats have been developed.

Well Known Text(WKT)

Well known text is a format defined by the Open Geospatial Consortium(OGC) and used for specifying geometrical object with vectors. WKT support multiple shape types like polygons, lines, multilines (several lines combined) and points[25]. The data format is presented in plain text, although it can easily be converted to its binary equivalent Well Known Binary (WKB). Coordinates are not limited to two dimensional space, so data stored in WKT can easily be extended to store altitude if needed. An example of a WKT entry is shown in listing 3.1.

Listing 3.2: Example of GML data.

```
<gml:Polygon>
   <gml:outerBoundaryIs>
      <gml:LinearRing>
         <gml:coordinates>0,0 100,0 100,100 0,100 0,0
         </gml:coordinates>
      </gml:LinearRing>
   </gml:outerBoundaryIs>
</gml:Polygon>
```

Listing 3.3: Example of KML data.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Placemark>
  <name>New York City</name>
  <description>New York City</description>
  <Point>
    <coordinates>-74.006393,40.714172,0</coordinates>
  </Point>
</Placemark>
</kml>
```

Geography Markup Language(GML)

Geography Markup Language is a XML based language and like WKT also defined by OGC[24]. While WKT is a more general language for all type of coordinates and vectors, GML is focused on modeling geographical data. GML consists of a large set of different types such as geometry, time, topology, direction, coordinate system and features. Since GML is XML-based a GML-file does not have to contain all of these object, only the ones relevant to ones application. An example of a GML entry is shown in listing 3.2.

Keyhole Markup Language (KML)

Like GML, the Keyhole Markup Language is also a XML based standard. It was first used by Google in its Google Earth application to symbolize locations. The main focus of KML is to easily visualize geometrical features, therefore is consist of much fewer types compared to GML. Unlike GML, KML only supports the use of WGS84 as a coordinate system[26]. An example of a KML entry is shown in listing 3.3.

Listing 3.4: Example of GeoJSON data.

```
{ "type": "Polygon",
  "coordinates": [
    [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
      [100.0, 1.0], [100.0, 0.0] ]
    ]
 }
```

GeoJSON

GeoJSON is geographical data format based on JavaScript Object Notation (JSON). It supports several shape types much like GML or WKT and it supports several coordinate systems, although WGS84 is the default choice[2]. The main advantage with using GeoJSON is JSON's compact size compared to XML-data. An example of a GeoJSON entry is shown in listing 3.4.

3.6. Data Storage

In a project like this, where large amount of spatial data need to be gathered and stored, a more advanced storage medium than an ordinary relational database should be considered. Common scenarios would involve finding out if geographical objects intersect, checking if a point is inside an area or finding the closest geographical object to a point. In a regular database this could be achieved by either storing all object coordinates as numeric values or by storing the string value of WKT/GML/KML/GeoJSON objects. Then one would have to fetch all coordinates, parse them, and compare each value to the current object with the appropriate function, like intersects() or isInside(). However, this approach does not scale very well when there are thousands of entries in the database.

3.6.1. GIS, OpenGIS and Spatial databases

A Geographic Information System (GIS) is a system specialized at storing, managing and accessing geographical data[11]. In a GIS system data can be selected by several geographical parameters, for example all objects within a radius of a point or all object intersecting a certain route.

A standardized way of realizing GIS systems is defined in several OpenGIS standards which are maintained by the OGC. The OpenGIS Simple Feature standard defines an interface for storing and accessing geospatial data in relational databases[25].

The standard describes how objects like points, lines, curves and polygons can be modeled and ways of querying data based on properties like if an object is within another object, if two objects intersects or by selecting all objects within a distances from a certain object.

Data can be inserted in the database by using standard formats like WKT or WKB. There are several existing projects implementing OpenGIS functionally to existing Database Management Systems (DBMS). For example PostGIS[2] is a project implementing the OpenGIS Simple Feature standard for PostgreSQL and SpatiaLite[3] provides geographical functionality for SQLite.

The main advantages of using a spatial database is the higher performance due to faster data selection and the ability to perform spatial operations at the database level instead of the program level. Spatial databases also reduces the programmers workload, by not having to implement spatial functions[4].

3.7. Presentation

The presentation layer is the last step before information reaches the user of the system. This section will describe how different parts of the presentation layer will function and what their respective role is in the NS-FRITS system.

3.7.1. Visual

The most basic form of presentation is visual presentation. The visual view of the NS-FRITS system varies because there will be different implementations for different roles. The drivers view of the NS-FRITS system is similar to a commercial GPS system. The main view will display the drivers current position on a map and a menu system will display all NS-FRITS dependent methods, such as search after POI, communicate with operator and plan route.

The operators view instead displays a list of all managed clients and their current status (online/offline/driving/etc). It should also consist of some kind of way to be able to create and assign jobs to drivers. To ease the planning job of the operator, the interface should also contain buttons to call NS-FRITS methods like search after POI and plan route.

3.7.2. Audible

Reading a text while driving can be both difficult and dangerous. The NS-FRITS system therefore offers the possibility to also present text data as sound. When a driver enters a zone where information is available, the drivers NS-FRITS client should play a sound file reading the text of the information. A driver might not be interested to get all types of information read to him, so it should be possible to only specify certain categories of interest.

[2] `http://www.postgis.org/`
[3] `http://www.gaia-gis.it/spatialite/`

3.7.3. Translation

One of the requirements of the system was that it should be multilingual, so an important topic is how text translation is handled. The project has defined three different types of translation:

- *Automatic translation*, where a server or a software receives a text and then translates it automatically.
- *Manual translation*, where the text or document is translated manually by an employed translator.
- *Partially automatic translation*, in this approach the text is first translated automatically, then verified manually by a translator to make sure that the text fulfills the required text standard. If the automatic translation fails to translate certain parts of the text, they are corrected by the manual translator.

Automatic and manual translation

Each of these translation methods have different uses. Which one to choose depends on the requirements of the text. Automatic translation works well with non-critical data that does not have any major impact on the driver. It is also an option for volatile data like road conditions and weather information, which might not be feasible to translate manually. Certain data can also be transformed into numerical data types, like for example temperature, speed limit or weight limit. Direct translation can then be avoided since the data only need to be presented in the correct language at the client side. The obvious advantage with automatic translation is the shorter translation time. Manual translation is still needed for more critical documents, like for example legal documents or contracts. Documents like customs forms[4] are not allowed to be translated automatically, since the correctness of these kind of documents need to be very high.

Partially automatic translation

The third option of partially automatic translation can be used in different ways, the first way is just as an aid to the translator. If the translating software has a certain grade of correctness, the time spent verifying and correcting an automatically translated text is lower than translating it manually from scratch. The other way of using partially automatic translation is to make the automatically translated documents available directly to the user, however, tagged in some way so the user understands that the text is an automatic translation and can not be completely trusted. The document is then sent to a translating/verification center where the text is queued for verification. The text is

[4] According to interviews with the Swedish Customs service in Svinesund, translation of customs forms is a strongly restricted procedure.

then verified by an employed translator and corrected if needed. The employee then sets the text tag to verified. It is also possible, if NS-FRITS would host a translation service, that the actor giving the translation job would have to accept the translation before it being published in NS-FRITS.

3.8. Communication protocols

The communication protocol is the protocol used on top of the communication technology. Many different communications protocols exist today, so the purpose of this section is to examine a few selected ones and their respective pros and cons. This information is then useful when selecting the ones to be used in the NS-FRITS prototype.

3.8.1. Web services

Web services are services that communicate over the HTTP protocol and data transmitted is usually formatted in XML, but JSON is also often used[13].

Because of the way the HTTP protocol works there is no native callbacks, but there exist some technologies that provide a similar functionality. HTTP was not designed to keep persistent states and requires all communication to be initialized from the client[7]. Without callbacks it is not possible for the server to push data to the clients.

The techniques involves different ways for the client to keep a connection to the server alive, for example by using a hidden iframe that the server can send data to in the form of Javascripts that gets parsed as they are received. Another way is to have a Javascript on the client that opens a connection to the server which the server keeps in an idle loop until some data is to be sent. When the data is received and the connection closed the Javascript will again open a new connection.

Almost all of these techniques are using some kind of functionality in the modern web browsers and thus are much harder to use in a custom application that only sends and receives custom data and not interpreting it as a webpage.

3.8.2. CORBA

The Common Object Request Broker Architecture is a communication middleware that allows applications to communicate independent of their programming language, hardware platform or operating system. It was introduced in 1997 by the Object Management Group (OMG) and provided programmers a tool for developing distributed systems with relative ease, but it has since then largely been replaced by, amongst others, web services[22].

Interfaces between CORBA clients and servers are defined in a Interface Definition File (IDL). The IDL is used to implement CORBA objects which are then used within the application as ordinary objects in the used language. An Object Request Broker (ORB) is used to handle the communication between the programming language's native objects and the CORBA objects and deal with the communication with other ORBs on other CORBA nodes[13].

A change in the IDL interface requires that all nodes in the system to be upgraded to support the new version[22].

3.8.3. XMPP

The Extensible Messaging and Presence Protocol (XMPP) defines a protocol for streaming XML data. It is the foundation for the Jabber/XMPP instant messaging (IM) service which is an open alternative to proprietary IM services as ICQ, Windows Live Messenger and Yahoo! Messenger. However, the XMPP protocol is not limited to providing IM services, it can also be used as a building block in a wide range of XML applications[44].

The protocol is designed to be a secure, decentralized, scalable and extensible way to exchange XML data[45]. In addition to the instant messaging service XMPP has been used in many areas such as game communication, voice over IP (VoIP) and social networking.

Instead of dealing with XML data as documents, as is the ordinary case, XMPP deals with XML data in the form of streams. A stream is a container for exchange of XML elements between two parties on the network. These XML elements are called XML stanzas, and the XMPP core specification defines three stanzas; `<message/>`, `<iq/>` and `<presence/>`[42]. The `<message/>` stanza is used to "push" information from one party to another. The `<iq/>` stanza is used for queries, similar to HTTP requests where one party makes a request and the other party sends a response. Finally the `<presence/>` is a form of broadcast mechanism which sends information to all subscribed parties. Usually used to send availability information.

The stanzas can be extended by introducing new XML namespaces. This is how the IM part is specified, the *Instant Messaging and Presence* extension[43] extends the three stanzas with more attributes and child elements to describe the IM data.

3.8.4. Custom protocol

Instead of selecting an existing communication protocol, one could instead define a new protocol specialized for the task. This approach has a number of advantages, it is possible to add specialized functionality not supported in existing protocol and the performance of the protocol can be increased by optimizing the data flow for the specific task. However, there are several down sides with this approach too. Existing standardized protocols are already well tested and evaluated so they are less likely to fail by design errors.

Well known protocols, like CORBA or XMPP, have already been implemented in several platforms and programming languages, so one could use existing libraries to ease development. This also makes the application easier to port to different platforms.

3.9. Data Providers - DATEX II

As an example of a data provider for NS-FRITS, the traffic data exchange standard DATEX II was chosen for further explanation.

DATEX II is a standard for traffic data exchange between different actors. It is being developed by the European Union and is in use at various locations in Europe. The original DATEX was first developed to provide information exchange between traffic management centers, traffic information centers and service providers[8]. The second version, DATEX II, has been extended to include all actors in the traffic and travel information sector. One of the purposes with DATEX II is to facilitate development of pan-European services providing road information directly to the users.

3.9.1. Technical Overview

The DATEX II architecture consists of three different components, the management subsystem, the publisher subsystem and the delivery subsystem[9]. The management subsystem is the component where management administrators handles information such as which type of data is being published or what subscriptions are allowed to use the system. The publisher subsystem is the part where data suppliers access to upload data. The delivery subsystem is the component responsible for providing data to clients. Only the delivery subsystem is specified by the DATEX II standard, the other components are up to the developer to implement.

DATEX II supports three different operating modes, publisher push periodic, publisher push on occurrence and client pull. In the publisher push modes, the publisher initiates data delivery in either a fixed periodic time or when data is changed. In client pull mode the client initialize the data retrieval and the publisher responds with the data. The data itself is transfered either with HTTP GET:s or published via Web services. The structure of the response messages are XML based, defined by a Web Service Description Language (WSDL) or an XML-schema.

3.9.2. Trafikverket, the Swedish National Road Administration

The Swedish National Road Administration Trafikverket, former Vägverket, has implemented DATEX II for publication of information about Swedish roads. Users can get information such as traffic conditions, road construction work, frost damaged roads and

road weather. The data is available in a web portal, for regular users[5], and as raw data requests for developers and professional users. Trafikverket's implementation of DATEX II has support for both PUSH and PULL for data retrieval.

[5] http://trafikinfo.vv.se/triniMenu/trinimenu.html?startmenu=1

4. System design

As a result of this project a design suggestion for the NS-FRITS system has been developed. This chapter will present and explain this design in detail.

4.1. Modules of the NS-FRITS system

The designed NS-FRITS system can be concluded in a number of interconnecting modules. This section will describe each module and their respective role. An overview of the designed system with modules and actors is illustrated in Figure 4.1.

4.1.1. NS-FRITS Core server

The NS-FRITS core server is the central server in NS-FRITS, it is responsible for the NS-FRITS data model, the NS-FRITS user base and the NS-FRITS API. Its basic purpose is to provide data to users via a defined API that should be supported by all NS-FRITS clients. Examples of API-calls include searching for a POI and getting information about a route. This will be explained further in section 4.4. The NS-FRITS core server is stateful, storing entries about each connected user. Examples of data stored for users are: Preferred languages, selected categories and stored route.

4.1.2. NS-FRITS Data provider

An NS-FRITS Data provider is a company or an organization providing data for the NS-FRITS system. When a data entry has been stored in NS-FRITS, it is directly available for users when searching after POIs or planning routes. Two main types of data providers has been identified in the NS-FRITS system, legacy providers and NS-FRITS providers.

Legacy providers

A legacy provider is an existing data provider, providing data in a format not compatible with NS-FRITS. These providers are already publishing data today accessible with some sort of API, for example a web service. An example of a legacy provider is providers of road and traffic information using the DATEX II format, for example the Swedish Road Administration.

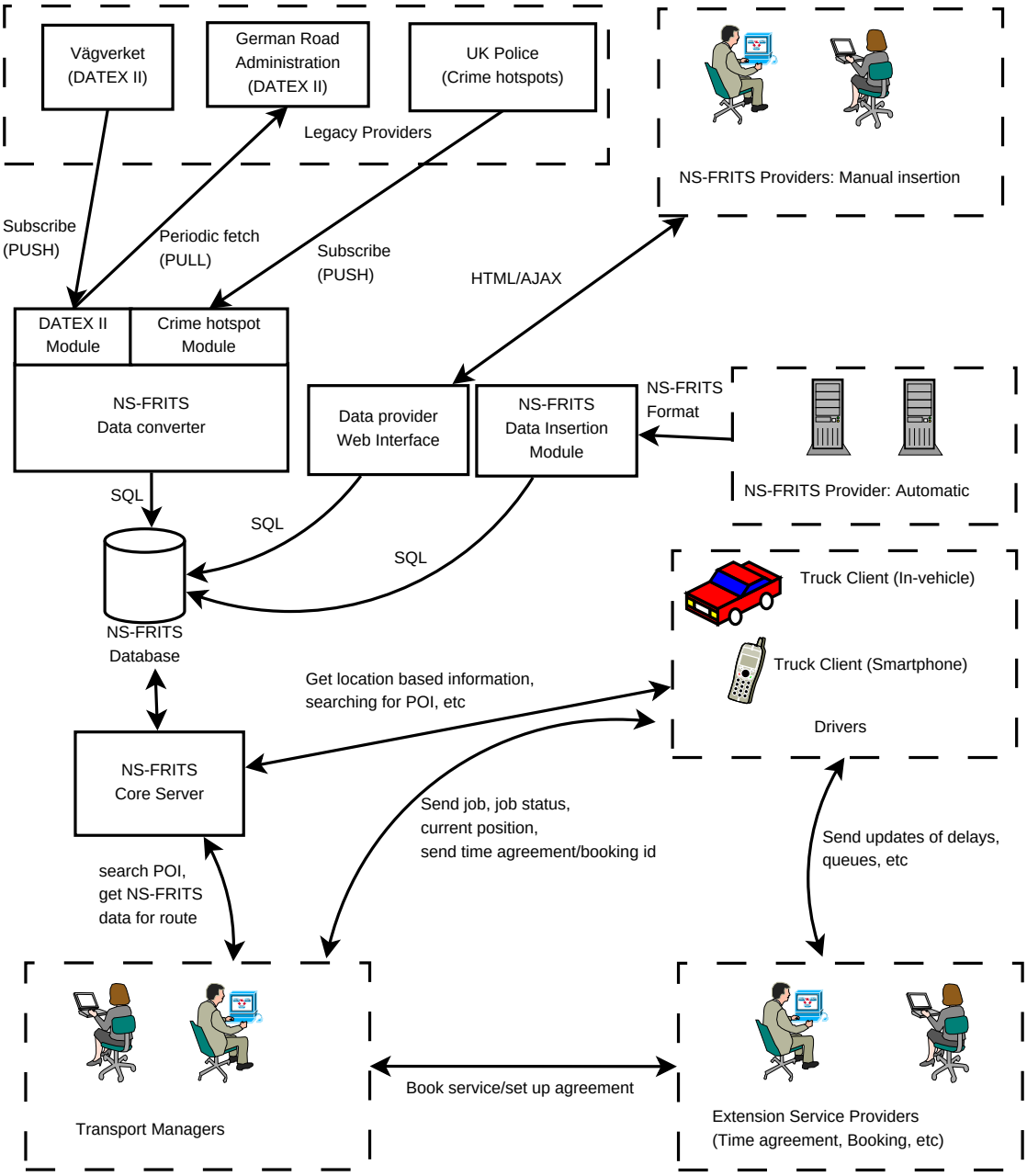

Figure 4.1.: Overview of the proposed NS-FRITS system design.

NS-FRITS will deal with legacy providers by periodically fetching data from them or subscribing to their PUSH-service. Incoming data will be converted into the NS-FRITS format through a NS-FRITS data converter module, the data will then be inserted into the NS-FRITS data model. The data converter module is a service that has specially written routines to convert the existing formats into the NS-FRITS format. The example

in Figure 4.2 shows how data from different legacy providers can be converted into NS-FRITS format.

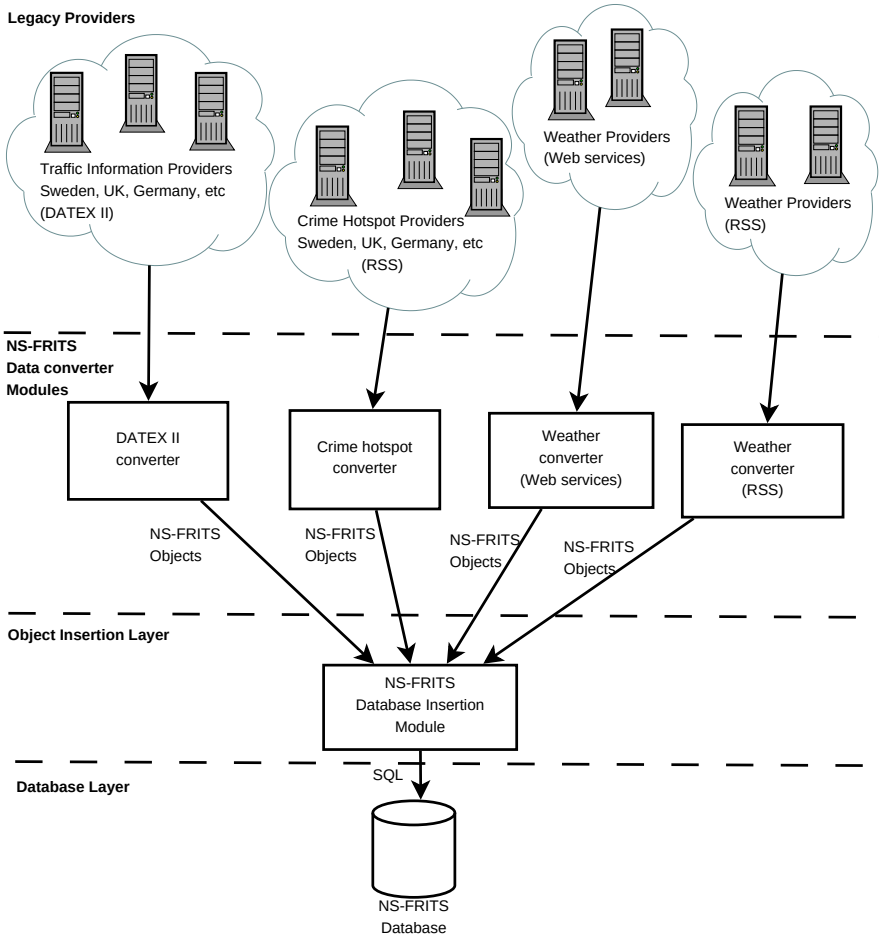

Figure 4.2.: This figure shows how legacy providers insert data into NS-FRITS.

NS-FRITS providers

An NS-FRITS provider is a provider that directly inserts data in the NS-FRITS data format. This could be done in different ways, either by manually entering the data through a web interface or by providing the data automatically through a service. An example of entering data manually could be a customs officer entering information about rules at certain customs stations through an application connected to the NS-FRITS database. An automatical data provider could for example be a road service company, providing data about road conditions through sensors.

An illustration of the workflow of NS-FRITS providers can be seen in Figure 4.3. Manual

providers enters data through a web interface connected to the NS-FRITS database. Automatic providers periodically sends new data to an NS-FRITS Data insert server, which then inserts the data into the NS-FRITS database. This data is then accessible by the NS-FRITS core server.

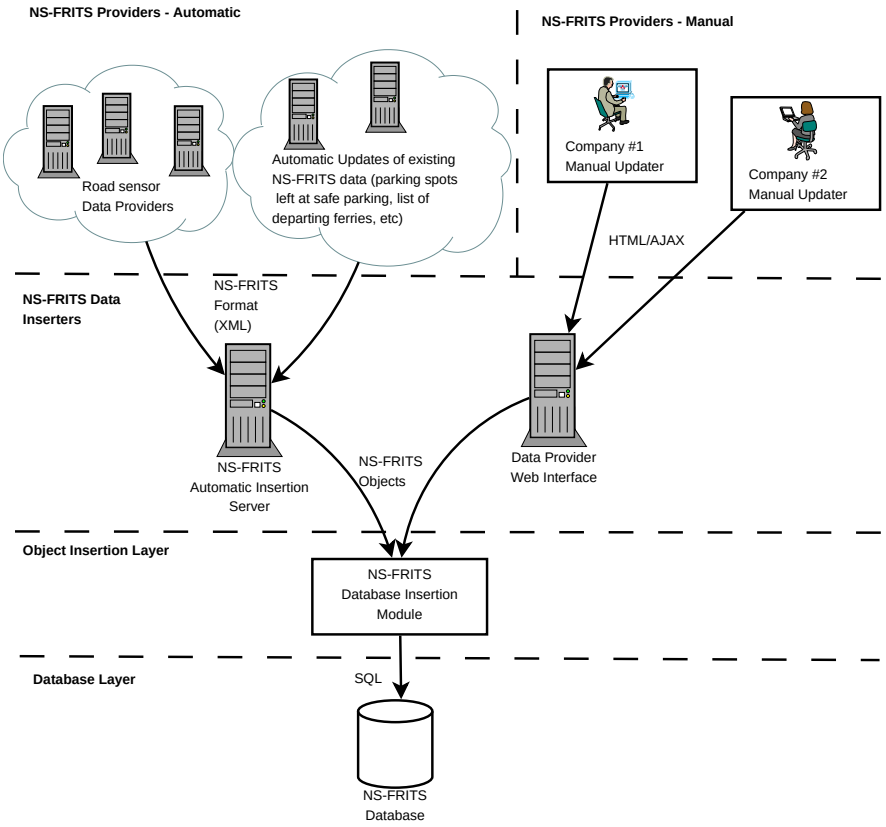

Figure 4.3.: Figure showing different NS-FRITS providers adding data into the system.

4.1.3. NS-FRITS extension

An NS-FRITS extension is an application that does not have to use the NS-FRITS database but resides in the same overlay network. Examples of these types of applications include fleet management systems, time agreement services or booking systems. These applications does not have to be hosted by NS-FRITS, instead anyone can host their own NS-FRITS extension. The idea of using an NS-FRITS extension instead of just using a separate application is that it can be integrated in the NS-FRITS Client (which will be explained in section 4.1.4). An NS-FRITS extension will also get the benefits attached to the usage of XMPP, like for example ability to reach users directly no matter where the user is and server caching of messages to offline users.

NS-FRITS extensions can generally be divided into two different groups, public extensions and private extensions.

Public extensions

Public extensions are like the name suggest, available for public use. Access to these extensions are shared through links stored in the NS-FRITS database. For example, the customs service at the border station in Svinesund might use a public extension to let trucks register their arrival and in case of queue times, send out updates to registered drivers. By then storing a link to the extensions attached to the "Svinesund Customs" object in the database, users searching for information about the Svinesund customs station will receive the information that Svinesund also contains a public extensions for queue updates. Users can then optionally choose to register their arrival to the extensions.

Private extensions

Private extensions are extensions that only a selected group of users can access, links to these extensions are not stored in the NS-FRITS database. Access to private extensions are usually gained by pre-configuring their address and protocol in the NS-FRITS client. An example of a private extensions is a fleet management system, i.e. an extensions that keeps track of trucks in a vehicle fleet. This type of extensions does not have any connection with the NS-FRITS core system, but can still benefit from running inside the NS-FRITS network. For example it can be run inside the same client application and the XMPP network makes it easy to maintain a direct connection to all trucks in the network.

4.1.4. NS-FRITS client

The NS-FRITS Client is the client that communicates with the NS-FRITS core server or an NS-FRITS extensions. This client can be deployed on different platforms depending on needs. For example a transport manager might run a desktop version, while a truck driver might run a version implemented on a smartphone or a mounted in-vehicle system. However, all clients should implement the NS-FRITS core server API and support a set of pre-defined protocols.

It is also assumed that mobile NS-FRITS clients have access to a GPS device to provide their current position. Although GPS is not a part of the NS-FRITS system, a mobile NS-FRITS client is assumed to have access to one, otherwise the NS-FRITS system would not be very useful since it depends on location based information.

Pre-defined protocols

Each NS-FRITS client should support a fixed group of pre-defined protocols, providing common functionality to users. One example of these pre-defined protocols are the time-agreement protocol mentioned in section 4.1.3. The reason NS-FRITS clients should support these pre-defined protocols is to make the deployment of NS-FRITS extensions easier. A provider can write applications following these pre-defined protocols and expect that they works with any NS-FRITS client. The relations between extensions and pre-defined protocols is shown in Figure 4.4.

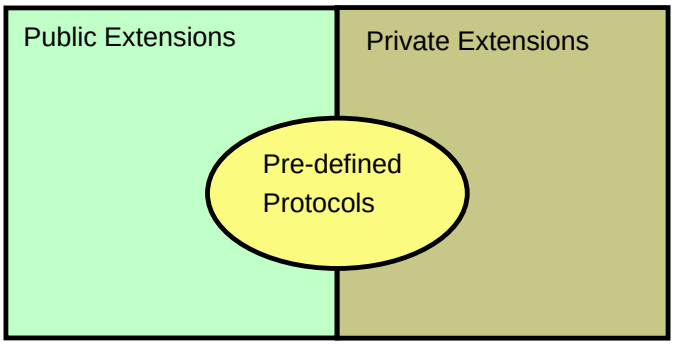

Figure 4.4.: Illustrates the relation between extensions and pre-defined protocols. An extension can either be a public extension or a private extension. If the extension uses a pre-defined protocol, it will function with every NS-FRITS client; otherwise the protocol for the extension need to be provided in some other way.

4.2. System design choices

The proposed design of the NS-FRITS system is a hybrid architecture based on the XMPP protocol. There were five main reasons why XMPP was preferred over its alternatives:

- Existing pre-defined support for different message types such as asynchronous messages(the `<message/>` stanza) and remote function calls(the `<iq/>` stanza).
- It solves the problem of not being able to reach trucks remotely. Trucks are always connected to the XMPP network, and in case of network problems, the XMPP

server can cache messages for offline trucks. In the case where the truck switches network, the truck just reconnects to the server.

- Not limited to one server. XMPP has excellent support for inter server communication. This allows fleet operators to use their own XMPP server in connection with the NS-FRITS platform.
- Possibility to see which users are online and connected to the NS-FRITS network with the presence information.
- Good extensibility, in addition to the possibility of defining your own protocol inside XMPP, XMPP have also defined a list of XMPP Extension Protocols (XEPs). These describe how different technologies such as SOAP, Publish-subscribe, RPC etc, can be implemented inside XMPP.

4.2.1. Low level communication

For high level communication XMPP is being used, which operates on top of TCP. The lower level communication requirements of the system are not as strict, any technology that can carry IP will work. However, most design choices were based on the assumption that telecommunication, such as GPRS, GSM or 3G, would be used in the majority of all cases.

4.2.2. Client to server communication

For client to server communication and vice versa, an XML protocol was designed. The designed protocol can be seen as a combination of XML-RPC and SOAP, but unlike XML-RPC the designed protocol supports custom tags[50]. This protocol is then used inside XMPP for transferring requests, responses and updates, see Figure 4.5 for how all layers interconnect. Data objects are transfered by first serializing the objects into XML and then sending them inside this protocol, for more information about the data structure see section 4.5.

4.2.3. Geographical data storage

For storing geographical data the Well Known Text(WKT) format was chosen. The three main reason for this decision are:

- It is a overall commonly used standard for simple coordinate data.
- Good support in spatial databases where WKT can be stored efficiently by converting it into its binary equivalent WKB. In this form it can also take advantage of spatial operations directly at database level.

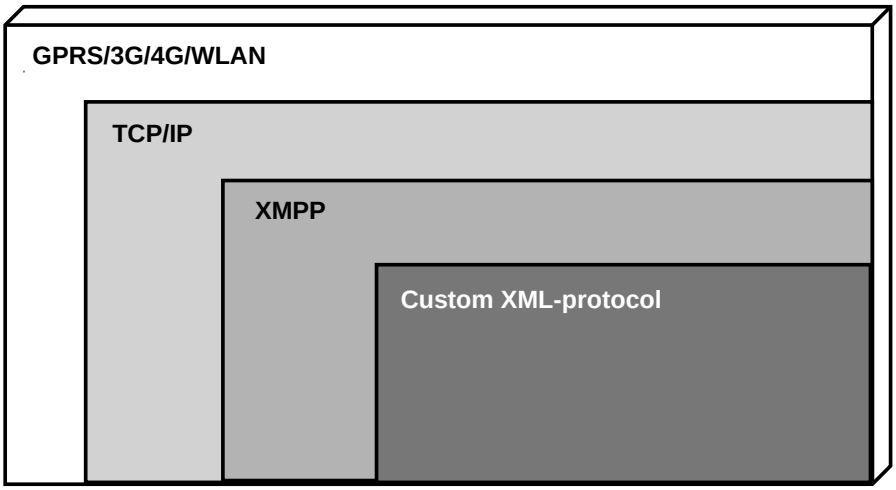

Figure 4.5.: Shows how different protocols and technologies are layered in the NS-FRITS design proposal.

- The extra functionality in KML and GML is not needed since a more complex data structure for information objects is needed.

4.3. The role of XMPP

As mentioned before, the communication protocol used in the NS-FRITS proposal is XMPP. The XMPP protocol is used for all client-to-server and client-to-application communication in NS-FRITS. This section will explain further how this is handled.

4.3.1. XMPP design

In this design, the NS-FRITS core server will act as a client to the XMPP server just like the other users of the system. It is possible to visualize the NS-FRITS core server as a bot to the XMPP server. The other users; truck drivers, transport managers, etc, will send updates and function calls to the NS-FRITS core server through the XMPP server. This architecture also allows direct communication between transport managers and truck drivers (see Figure 4.6).

4.3.2. External servers

This first simple model might not be enough for a real system, but since the XMPP protocol supports interaction with multiple servers, it is very easy to extend. For example, communication between a transport manager and its trucks does not involve the central

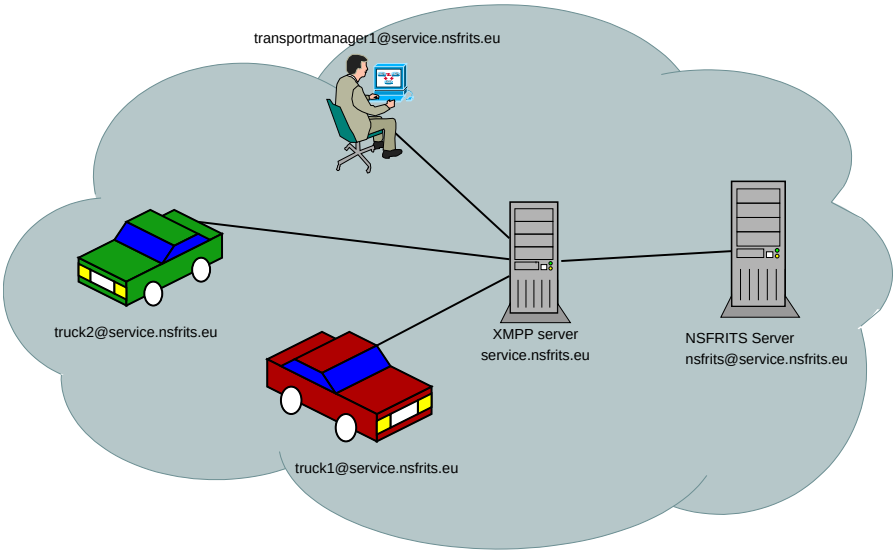

Figure 4.6.: Simple overview of the XMPP architecture for the NS-FRITS system

NS-FRITS server. Therefore it is possible for each freight company to use their own XMPP server (see Figure 4.7). In this way, internal data stays on the companies own XMPP network, while request to the NS-FRITS core system is passed on to the central network. There are two main benefits by separating a network like this. First, the security is improved, by involving less parts and keeping the information as local as possible the risk of security breaches decreases. Secondly, by splitting up the users so not all data have to pass through the NS-FRITS system, the load of the central NS-FRITS server(s) will be reduced and the central XMPP network will scale better with many users.

4.4. NS-FRITS methods and functionality

The NS-FRITS server supports several method calls through a defined API. This section will explain the functionality of the NS-FRITS API and how each method works.

4.4.1. Methods controlling user data

The first methods to be explained are the ones controlling information stored to a particular user. There are five methods for NS-FRITS client to control what information the NS-FRITS server knows about them:

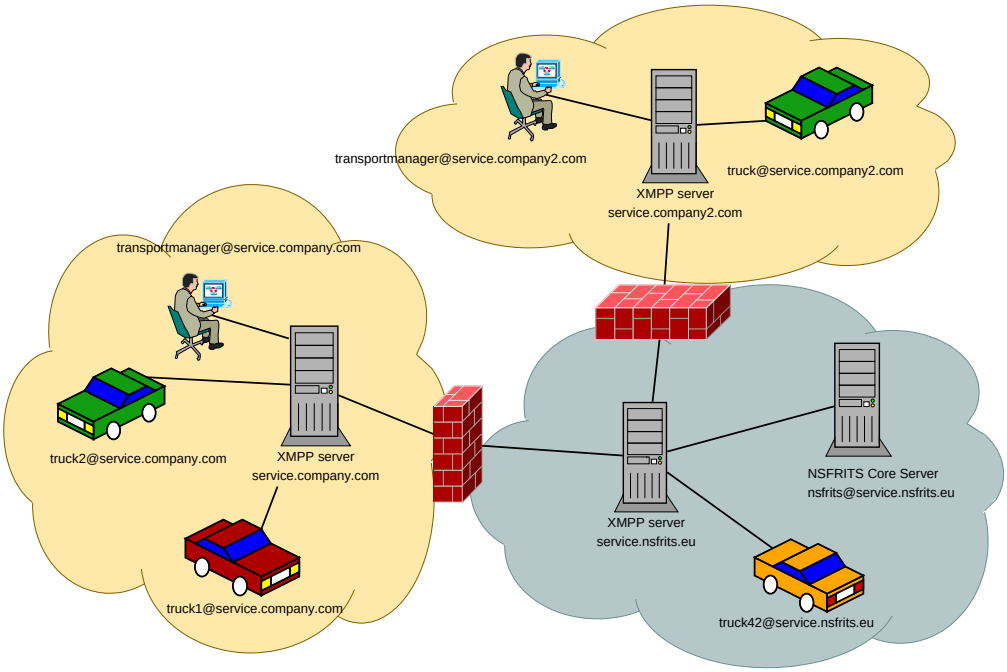

Figure 4.7.: XMPP architecture involving multiple servers, trucks communicating with transport managers stay inside their respective network. The transport companies networks are isolated from each other.

Get and set categories

By using these methods the client can inform the server of what data categories the client is interested in. Each data provider belongs to one or more categories, for example the Swedish Road Administration and the German Road Administration might belong to the "Road Updates" category. By calling the "Get categories" command the server returns a list of all available categories and a description to each one. The client can then call the set categories command and as an argument include a list of all categories the client is interested in. When the client later asks the server for information for a certain route or coordinate, it will only get information in the previously set categories. Subsequent calls of the set categories command will overwrite the previous values.

Set languages

The third available function for controlling user data, is the set language method. By setting a list of known languages, the client can control what languages the server will send information in. Languages are encoded according to the international standard for language codes, ISO 639-1[18], so there is no need for the server to provide the client with all available languages with a get languages method.

As an argument to the set languages method, the client sends a list of what languages it is interested of receiving data in. The list should be ordered by the highest priority language first. After languages has been set, the client will only receive data in the highest available language in the priority list. If the information is not available in any of the languages the client has set, it will receive the data in one of the available language as default.

For example, a Polish driver sets his languages in the following order, Polish, German and English. When his NS-FRITS client asks the central NS-FRITS system for data, it will only receive data in one of those language if available. If the data is available in Polish, the server will only send the polish version of the data object. If the data is available in German and English, the server will send the German object since its priority is higher set than English. If the data is not available in any of those three language, the server will send whatever version it has.

Set a route

The set route method is used by clients to inform the server of what route the client is planning to use. This method is tightly bound to the get information for route function which will be explained in section 4.4.2. The argument for this function is a WKT Linestring of all coordinate along the route.

Clearing of history

The clear history command removes the history of InfoObjects already sent to the client. The storage of sent objects is explained more under section 4.4.5.

4.4.2. Getting information for a route

The command for getting information for a route requires no arguments, instead it relies on that the user has uploaded a route prior to using this command. When this command is called, the server first checks what categories the user is interested in (given that the user has run the set categories command at least one time before), then retrieves all information objects along the previously set route. The server will then filter out objects that it already have sent to the client. Finally the server will fetch the description of each object in the highest available language in the users language priority list. The list of objects will then be sent to the client.

If the client calls the get information for route more than one time, it will only get new or updated objects in the latter calls, see section 4.4.5 for more about storage of sent objects.

4.4.3. Getting information for coordinate

The get information for coordinate method is similar to the get information for route method with the difference that the client instead sends a coordinate as method argument. Upon reception the server will fetch all object that the specified coordinate is within and return them to the client. Like the get information for route method this method will not resend already sent objects unless they have been changed or the history has been cleared. This method can be called periodically by driving trucks to receive local data from NS-FRITS if no route has been set.

4.4.4. Searching for a POI

The search for Point of Interest (POI) method is used for searching for a particular POI on the map. This method has two difference from the get information for coordinate method, first, the category is specified as an argument to the function and secondly, searchPOI is not limited by a distance radius; it simply retrieves the closest POI, no matter the distance from the given point. The main intended usage of this function is to find a location in a wanted category at a certain spot, like for example the closest safe parking spot. Another difference compared to the get information for coordinate method is that the wanted number of results is passed as an argument. This can be used if a driver for example want to find the five closest safe parking spots to a location where he/she plans to stay.

4.4.5. Storing of sent objects

Since data sent over the telecommunications network can be both expensive and slow, unnecessary data traffic want to be avoided. For this reason a functionality to avoid resending of objects has been designed. Whenever an object is sent through the get information for route or get information for point method, the objects id and its sequence id is also added to a user unique list of sent objects. If the user later calls any of these functions again, the server will not resend any object that are already sent, unless the object has been updated with a newer version(i.e its sequence number has been changed). When a client is reset, this list need to be cleared with the clear history command explained in section 4.4.1

4.4.6. Subscribing to an information object

The subscribe to information object method is used by clients that want to receive updates when a certain information object has changed. For example, a safe parking company might use an information object to store how many parking spots is left at a certain location. A truck might then subscribe to the information object bound to several safe parking locations. Whenever any of these objects are changed, updates will

be sent out to all subscribing clients. The method only have one argument, the id of the object the client want to subscribe to, this id can be retrieved by earlier running the get information for point/route or search POI command. When the object is updated, the server sends an asynchronous message to the client containing the update object, for more about asynchronous messages see section 4.5.3.

4.4.7. The NS-FRITS alert system

The NS-FRITS alert system is an optional add-on to the standard NS-FRITS service, clients can participate in it by sending out coordinates about their current position periodically. The purpose of the NS-FRITS alert system is to keep clients updated of local alarms, important messages and accidents. Whenever an event occurs, the NS-FRITS system will locate all trucks in a radius of the event and sent out a message to each of them containing the event message.

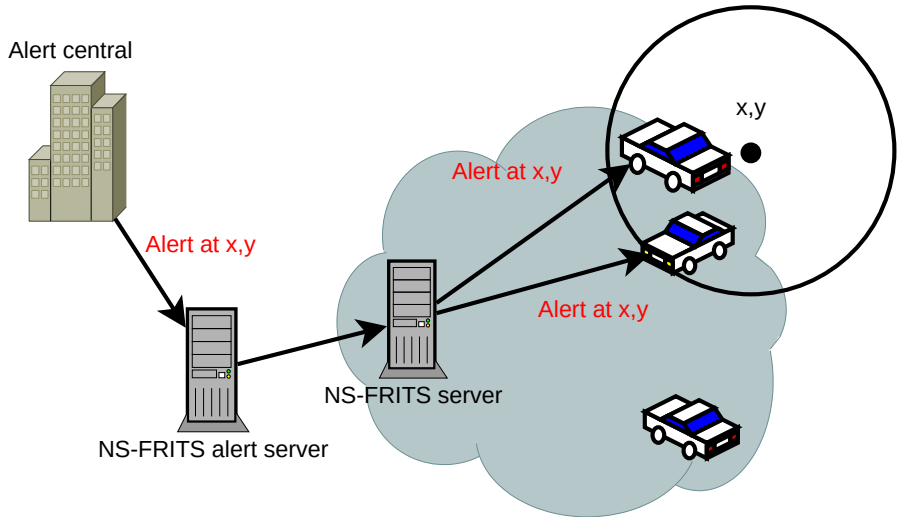

Figure 4.8.: Illustration of how distributions of alerts from an external authority is handled. Only the trucks in the proximity of the alert position is noticed.

Alert can come from different sources, either centrally from an authority or an information central(Figure 4.8), or from other trucks(Figure 4.9). By sending a special alert message to NS-FRITS, trucks can inform other trucks in the area of an accident or an alarm. It can also be connected to the vehicle's crash sensors, if it is equipped with such system, and automaticaly send alerts to the main NS-FRITS system.

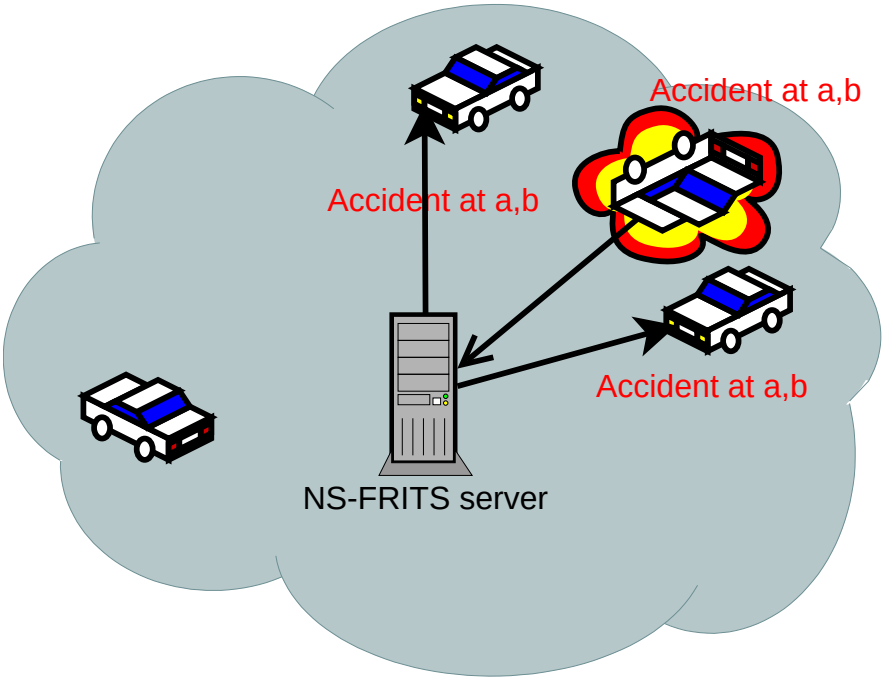

Figure 4.9.: Clients reporting directly to the NS-FRITS server of an accident, the alert is sent to nearby trucks.

4.5. Data protocol

The data protocol is the protocol used inside XMPP for function calls and object transfers. There are several ways of transferring data over the network, which were explained in more detail in section 3.8. For function calls, the XMPP stanza `<iq/>` is being used, as explained before this is the XMPP feature for standard querying mechanism.

The protocol selected was a custom XML-based protocol. The main advantages with specifying an XML-based protocol is platform independence, easy to implement and high readability. At the top level of iq stanza there can be one of either tags, `<methodcall>` or `<methodresponse>`. The methodcall tag is used for function calls, while the methodresponse tag is used for function responses.

4.5.1. Method calls

The methodcall tag is also accompanied by an `<arguments>` tag, which contains all arguments passed to the method as a map. When the driver executes a command the NS-FRITS client converts this command into its corresponding XML-format. This command is then inserted into a XMPP message in an iq stanza and sent over the network. See Listing 4.1 for examples of how a method response might look like.

Listing 4.1: Example of a method call to NS-FRITS, this function returns all InfoObjects inside the coordinate of the argument

```
<iq id='qxmpp10' to='nsfrits@VTECW389.vcn.ds.volvo.net/nsfrits' type='get
    '>
   <query xmlns="nsfrits:iq">
      <methodcall>getInfoForPoint</methodcall>
      <arguments>
         <argument>
            <key>point</key>
            <value>POINT(11.878967 57.737883)</value>
         </argument>
      </arguments>
   </query>
</iq>
```

When the packet reaches the server, the server will parse the content of the tags and execute the called function. When the function returns, the result is placed inside a methodresponse tag which is placed inside an iq result packet. The packet is then sent back to the client.

4.5.2. Method responses

The content of the methodresponse tag is either empty (for a void function) or a list of InfoObjects, which is the main data type. Each InfoObject corresponds to a certain location on the map and contains information such as coordinates, descriptions and links to attachments. The structure of InfoObjects are explained in more detail in section 4.6. When a function is called through a methodcall tag, the server will select the appropriate InfoObjects from the database, convert them into XML-form then put them inside the methodresponse tag.

4.5.3. Updates and Asynchronous messages

The other main functionality supported by the server is the possibility to send updates and alert to clients asynchronously, i.e the server does not expect any response. One of the functions available through the NS-FRITS API is the subscribe-to-InfoObject method. This function is called with an InfoObject id as an argument, and in case that particular InfoObject is modified by the provider, a update message containing the new InfoObject is sent to the respective client. This update message type is also used the other way around, where clients are sending updates about their current position to the server, which is used in the NS-FRITS alert system (see section 4.4.7).

Unlike the previously mentioned method calls and response, the update message type uses the `<message/>` stanza instead of the `<iq/>` stanza. The message stanza is XMPP's

Listing 4.2: Example of an asynchronous update sent by the server. The content of the InfoObject is left out.

```
<message id='Frmwt-10' to='truck1@VTECW389.vcn.ds.volvo.net/nsfrits'>
  <updates xmlns='nsfrits:async'>
    <InfoObject>
      ...
    </InfoObject>
  </updates>
</message>
```

solution to asynchronous messages type, and it provides the ability for the server to cache messages if the client is offline. The `<message/>` stanza is used by extending it with an extension for updates (see Listing 4.2 for an example).

4.6. NS-FRITS data model

This section will explain in more detail how the NS-FRITS data model looks like and how data is represented in it. An overview of the NS-FRITS data model is presented in Figure 4.10.

4.6.1. InfoObject

The main object used for representation of location based information is the InfoObject. Each InfoObject is mapped to one Location object, which symbolize the place where the InfoObject's data is relevant, and one DataNode which is the object where the information itself is stored. Further more, an InfoObject contains two more values, Validity and Sequence Id.

Validity

The InfoObject has a certain time interval when it is valid, which is implemented by using two fields, ValidFrom and ValidTo. For example a road accident might have a validity of only a few hours, while a text describing customs regulation perhaps have a validity of several years. The ValidFrom field also makes it possible for data providers to enter data which will not be valid until a certain date or amount of time has passed, for example new rules or regulations. When the NS-FRITS server sends out data to clients, it will ignore objects outside the valid time frame.

Sequence Id

Inside the InfoObject there is also a field called sequenceid, this fields purpose is to differentiate different versions of the same InfoObject. If a data provider updates an existing InfoObject in any way, this counter is incremented. When a client fetches a certain InfoObject from the server, the server stores which version number of the object was sent. If the client later tries to fetch the same object, the server compares the InfoObject's current sequenceid with the sequenceid stored for that particular client. If the current InfoObject has a larger sequenceid, the new version of the object is sent to the client, otherwise the client already have the latest version so the InfoObject is not included in the response.

4.6.2. DataNode

Each InfoObject is mapped to one DataNode, where the information itself is stored. The information in a DataNode is stored as a data tree, this is implemented by storing a field in each DataNode called children. The children field links to zero or more other DataNodes. The root element of the tree is then stored in the InfoObject, so when an InfoObject is fetched, the tree can be constructed by recursively fetching by all childrens. The DataNode tree structure is illustrated in Figure 4.11.

This tree approach also make it possible to reuse the same DataNode in several InfoObjects. For example the customs service might want to upload a form for declaration of goods into the NS-FRITS system. Since they want this form to be presented at every customs station, they only need to create one DataNode where they store the form and then link to this DataNode form the respective tree of each customs station.

Information can be stored in DataNodes in three different forms, Descriptions, Attachments and ParameterData. Attachment and ParameterData will be explained is this section, Description will be explained in section 4.6.4.

Attachment

The Attachment data type is used for the purpose of attaching files such as images, PDF documents or videos to DataNodes. The file itself is not stored in the Attachment object, instead only an URL link to its location is stored. The reason for doing this is to save unnecessary data traffic. When a driver receives InfoObjects for a planned route, he/she will probably only be interested in a small amount of all attached files, so instead files are fetched when requested.

In addition the URL link to the file, the Attachment object contains three more fields, mime-type[1], size and language. The mime-type contains what sort of content type the

[1]See more at `http://www.iana.org/assignments/media-types/index.html`

file has, the size is an integer containing the number of bytes in the file and the language what language the file is in. The reason for storing the file size is so that clients can see how large a file is before deciding to fetch it. If a DataNode contains several Attachments in different languages, the server will only use the link in the highest preferred available language to clients (See more about client language preferences in section 4.4.1). There is also a special language key word, "all", which represents that the Attachment is language independent.

ParameterData

ParameterData is used for sending out specialized language independent information such as temperature, speed limit, length or maximum weight. Except for the parameter type, the ParameterData also contains a field for the value of the parameter. The advantages of using ParameterData instead of one or several Descriptions is except for language independency, the possibility for clients to display this field in a special ways. For example, if a client receives an InfoObject containing an updated speed limit, it might pop up on the screen with a special icon with the new speed limit value. Each DataNode can contain zero or more ParameterData objects.

4.6.3. Location

The Location object is one-to-one mapped to the InfoObject, each Location object can contain four fields for representing where the InfoObject is of use, point, polygon, place and road. The most basic field is the point field, it represents a location on the map with a latitude and longitude coordinate in the WKT format. The polygon field represents what area a driver should receive notifications of the InfoObject. The value is stored as a WKT polygon, which is simply a list of coordinates together representing a polygon.

Each Location can also have a place and a road attached to it. A place is a general term used for symbolizing locations that are too complex to describe with geometrical objects, it is stored as a string. This type could for example be used for countries: Instead of sending a large polygon representing each edge of the country border, the place field can instead be used, containing the name of the country. The client can then find out what country it is in by using some sort of external service, like for example the GSM-network. A road is just a string describing which road the information is relevant in. This could be used in combination with a polygon to describe certain strips of a road.

4.6.4. Description

The Description object is the general object used for descriptions, both DataNodes and Categories can have one or more descriptions. Each Description consists of a title, text, language and sound URL. The title is the description of the description object itself,

while the text is the message of the object. Since DataNodes and Categories can have several Descriptions, it allows them to store one description for each language, which is stored in the language field encoded in the ISO 639-1 standard. Reading a text can be difficult and even dangerous while driving at the same time, therefore each Description can have a sound file attached to it. The sound URL field is an optional field that stores a link to a sound file that the client should load when opening the message. This sound file should preferably contain a voice reading the description text in its specified language. It can either be recorded manually by a person, or generated by a text-to-speech software

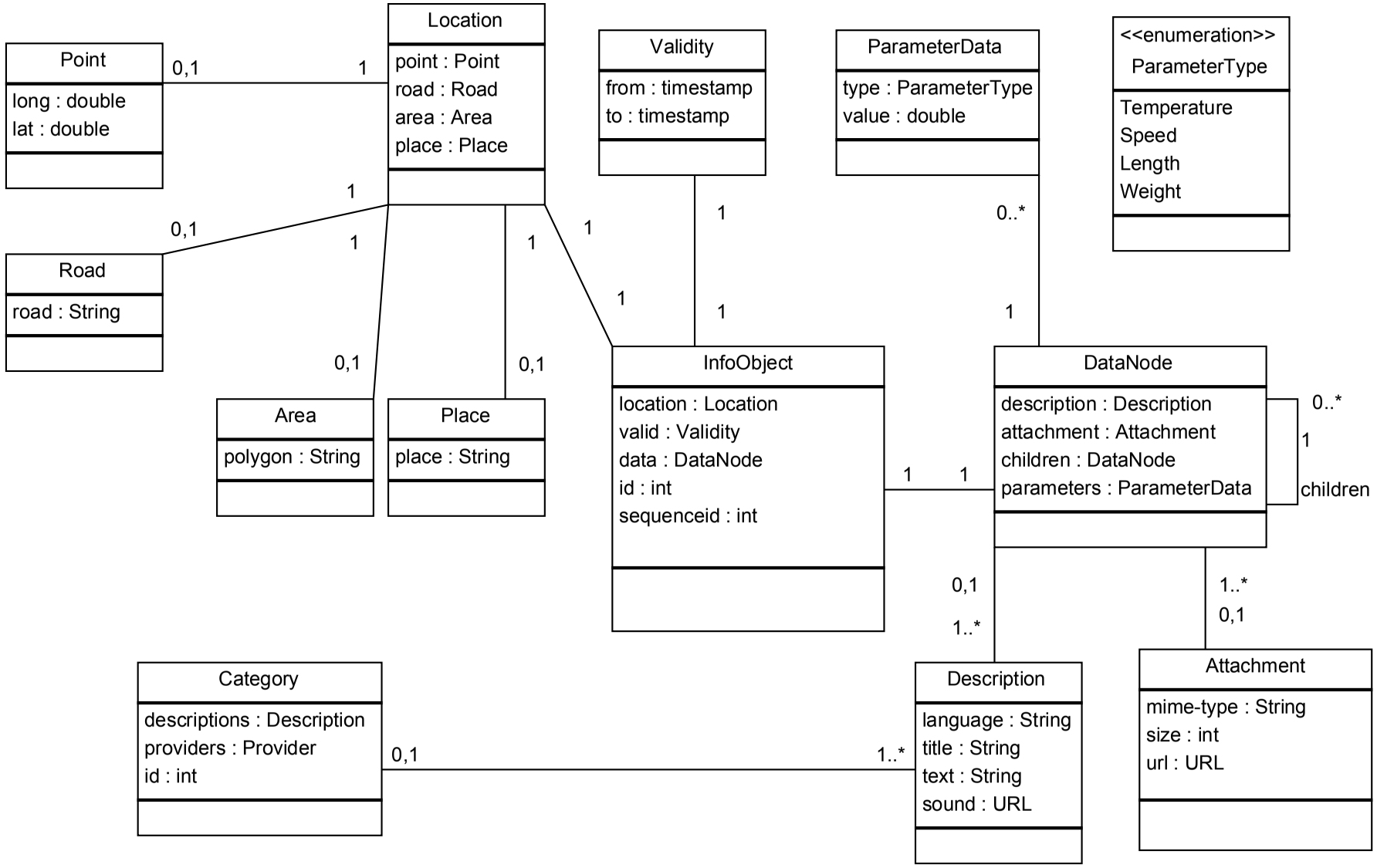

Figure 4.10.: Overview of the NS-FRITS data model.

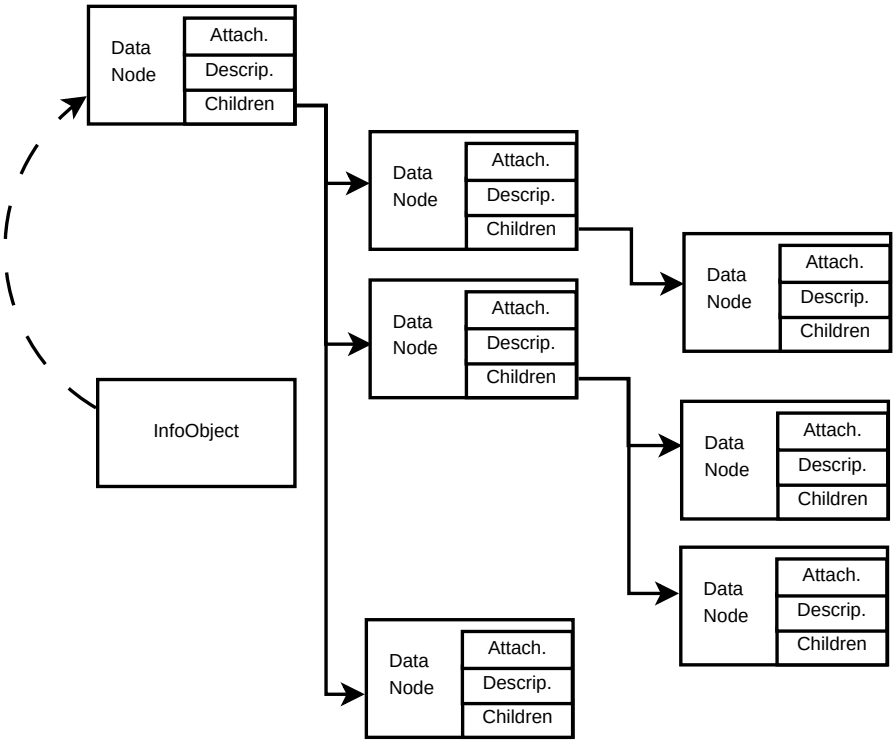

Figure 4.11.: A tree of DataNodes, each DataNode can have several children DataNodes, together building a tree. Each InfoObject is linked to one DataNode corresponding to the root of the tree.

5. Prototype implementation

This chapter describes what has been implemented in the prototype implementation of the NS-FRITS system and what technologies the implementation relies on.

5.1. Overview

Six different applications have been developed for the NS-FRITS prototype:

- *NS-FRITS Client application*, this is a desktop application that simulates a truck on a road and its in-vehicle unit. This application communicates with the NS-FRITS core server through the XMPP protocol.
- *NS-FRITS core server*, the NS-FRITS core server communicates with NS-FRITS clients through an XMPP-server. This applications purpose is to provide NS-FRITS data to clients through a common API.
- *Provider interface*, this application provides a user interface to the NS-FRITS database. An administrator can via a web application add and remove locations and points of interest. To each location, the administrator can add text, pictures, documents and audio files in different languages. The main role of this application is to simulate content providers.
- *DATEX II Parser*, an application for converting DATEX II data into the NS-FRITS format has been developed. This application loads DATEX II XML-files and inserts their content into the NS-FRITS database.
- *Time agreement application*, is an NS-FRITS extension that has been developed to show case external communication over XMPP. The application developed is based on a simple time agreement protocol. This protocol is used by trucks to register an estimated time of arrival and send messages to the server in case the truck gets delayed. The server operator will be able to see all registered clients on screen, and in case of delays such as queues, the operator can send messages to them. This time agreement protocol has been proposed to be one of the NS-FRITS systems pre-defined protocol (See more in section 4.1.4).
- *Fleet management/dispatcher system*, a very basic fleet management system that can assign orders to trucks and get updated information about the trucks current location and status. It is not part of the NS-FRITS protocol and is only used

to show the possibility of implementing a fleet management system using the NS-FRITS XMPP network. It can use the information provided by the NS-FRITS system to help the dispatcher make a fast and safe route for its trucks.

5.1.1. XMPP server

The XMPP server used in this implementation is Ejabberd from ProcessOne, but any compliant XMPP server would work. Ejabberd is a cross-platform open-source XMPP server written in Erlang and is well known for its good performance and scalability[45]. Other XMPP server alternatives include jabberd2 [1] and Openfire [2].

5.2. Truck client

5.2.1. Overview

The truck client implementation is a standalone PC application written in C++ with cross-platform libraries making it easy to port to different platforms. This implementation is not meant to be used in an actual truck, but used as a demonstration of the capabilities of the system.

The implementation is built on the framework built for the L-ViS application. In addition to this framework a group of other libraries are used to provide additional functionality. This section describes the L-ViS framework and the additional libraries.

In the main window, see Figure 5.1, there is a zoomable and scrollable map with an icon showing the current location of the vehicle. The maps used are provided by OpenStreetMap[3] which are licensed under the Creative Commons Attribution-ShareAlike 2.0 licence (CC-BY-SA)[4]. The vehicle can be moved around on the map by dragging and dropping it on the desired location.

From here the user can choose the desired languages and information categories, see Figure 5.2, which decides which data the server will send back to the client. Information can be requested either by requesting information close to the current location or by uploading a route and requesting all points along that route.

All nearby information points are shown in a list, and from which the user can choose to get more information on each item. The extended information is shown in a new window, see Figure 5.3. The information can be in many forms, for example as text, images, PDF-documents or web links. The text descriptions can be played out through a speaker if a voice file has been attached to the description. The built-in web browser,

[1] `http://jabberd2.xiaoka.com/`
[2] `http://www.igniterealtime.org/projects/openfire/`
[3] `http://www.openstreetmap.org/`
[4] `http://creativecommons.org/licenses/by-sa/2.0/`

Figure 5.4 is used to reach external web applications so that already existing services can be reached and used without any modifications on the existing system.

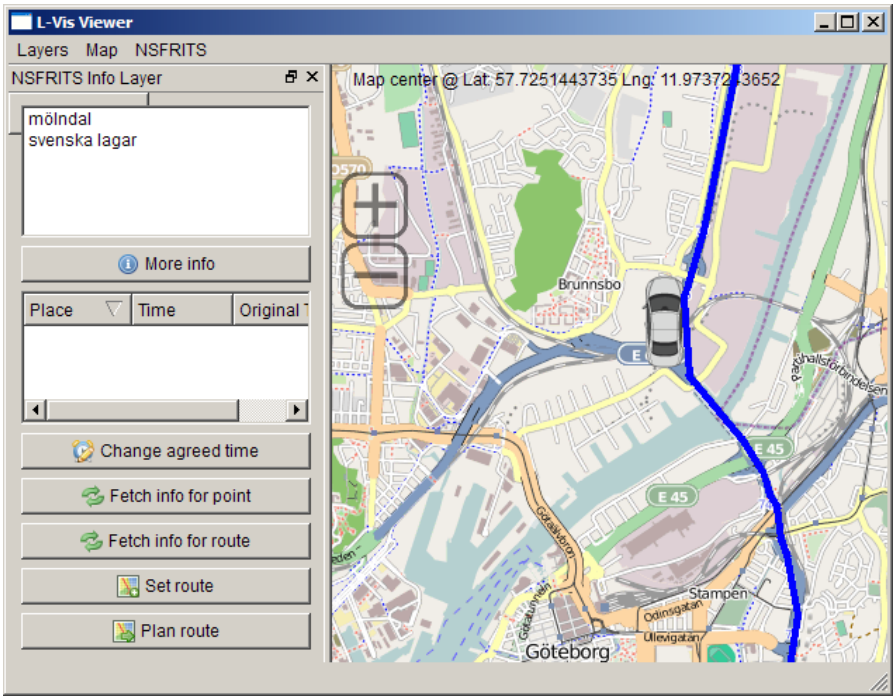

Figure 5.1.: Main window of the PC-based truck client. Visible is the current position of the vehicle and a planned route. In the information overview table the currently nearby information points are visible.

5.2.2. Libraries

L-ViS - LDM Visualization System

The LDM Visualization System is a system for visualization of global state information from CVIS/Safespot systems[31]. It consists of three parts, a state reporting application, a server and a visualization application.

The visualization part of the system provides a solid framework for development of applications that need to render maps and information related to coordinates. It is written in the language C++ for performance and library support reasons. Libraries used, in addition to the C++ standard library, include Qt[5] for GUI, GEOS[6] for spatial opera-

[5]Qt - Cross-platform application and UI framework. `http://qt.nokia.com/`
[6]GEOS - Geometry Engine, Open Source. `http://geos.osgeo.org/`

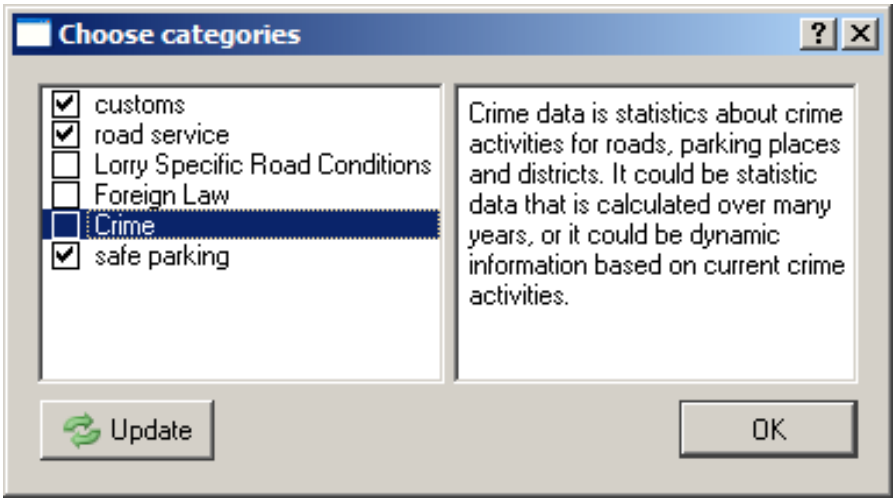

Figure 5.2.: The window were the user can choose the wanted categories. All supported categories are fetched from the NSFRIS system when the update button is pressed. The new categories are then uploaded to the users account on the server.

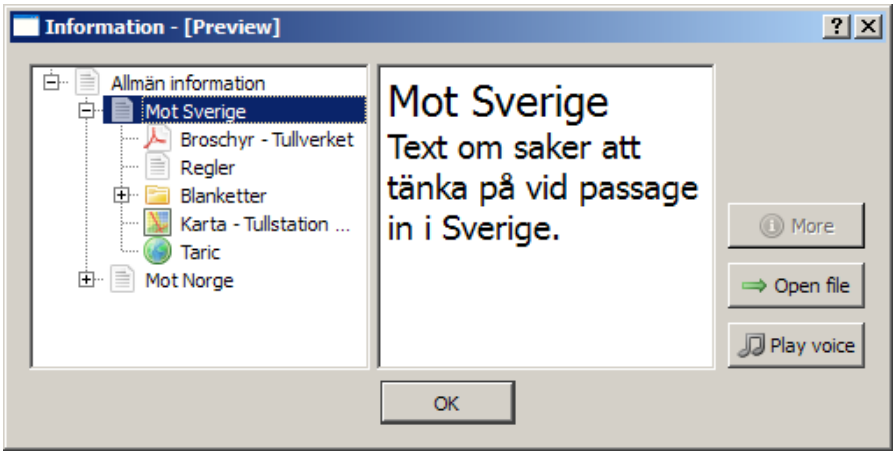

Figure 5.3.: When the user wants to get more information about an information point this window is shown. Texts can be read out if a voice file as been attached, and attached documents can be opened and shown to the user.

Figure 5.4.: A link to a web service provided by the Swedish customs has been opened in the built in web browser. By allowing links to external web pages already existing systems can be easily reached without any modifications.

tions, PROJ.4[7] for transformation of coordinate systems and Boost[8] for shared, reference counting pointers.

All the libraries used are platform independent and no code in the visualization application itself is platform dependent and is known to compile on both Microsoft Windows XP and Linux according to the developers.

QXmpp

QXmpp is a cross-platform XMPP library for C++ built on Qt. It offers support for the basic XMPP concepts and some of the many extensions for the protocol. The library deals with the XMPP communication behind the scenes and transforms incoming XMPP packets into objects that are sent to the main application.

Other libraries considered were libstrophe, iksemel and Iris. QXmpp was chosen primarily because of the suitable license (GNU LGPL) and the fact it offered easy integration into the existing Qt environment.

[7] PROJ.4 - Cartographic Projections Library. `http://proj.osgeo.org/`
[8] Boost C++ libraries. `http://www.boost.org/`

Simple DirectMedia Layer (SDL)

The SDL library is used to play the sound and voice files. The core SDL library together with the extensions SDL_sound and SDL_mixer form an easy to use package for playing audio files encoded in different codecs. The library is cross-platform and precompiled binaries exist for many platforms. Both the main library and the extensions are licensed under the GNU LGPL license.

Other considered sound libraries were PortAudio, OpenAL and Phonon.

Speex

As codec for the audio files the Speex codec was chosen. It is an open and royalty free format for voice audio and provides very good performance at low bitrates.

Webkit

Webkit is a web browser engine used by some of the major web browsers such as Google Chrome[20] and Apple Safari[1]. It is available under a mix of the LGPL and BSD licenses. Figure 5.4 shows the Webkit library as used in the NS-FRITS client application.

5.3. Fleet management

The fleet management application is also built around the L-ViS framework and uses the same libraries as the truck client. It allows the operator to see the current location and status of the trucks and assign new orders to them. Information from the NS-FRITS system is used to allow the operator to make decisions about the truck routes.

The fleet operator can also in beforehand set up time agreements to external actors and provide these to the truck. When an order gets assigned to a truck the driver gets a notification and responds by either accepting or rejecting the order. If accepted the truck will load the new route, upload it to NSFRITS and get the associated information points along the route.

During the trip the truck will send its coordinates back to the fleet management system and at the end of the drive reply back that the order has been completed.

5.4. NS-FRITS backend applications

The applications connected to the server side of NS-FRITS can be summarized under the umbrella term NS-FRITS backend applications. This consists of four different ap-

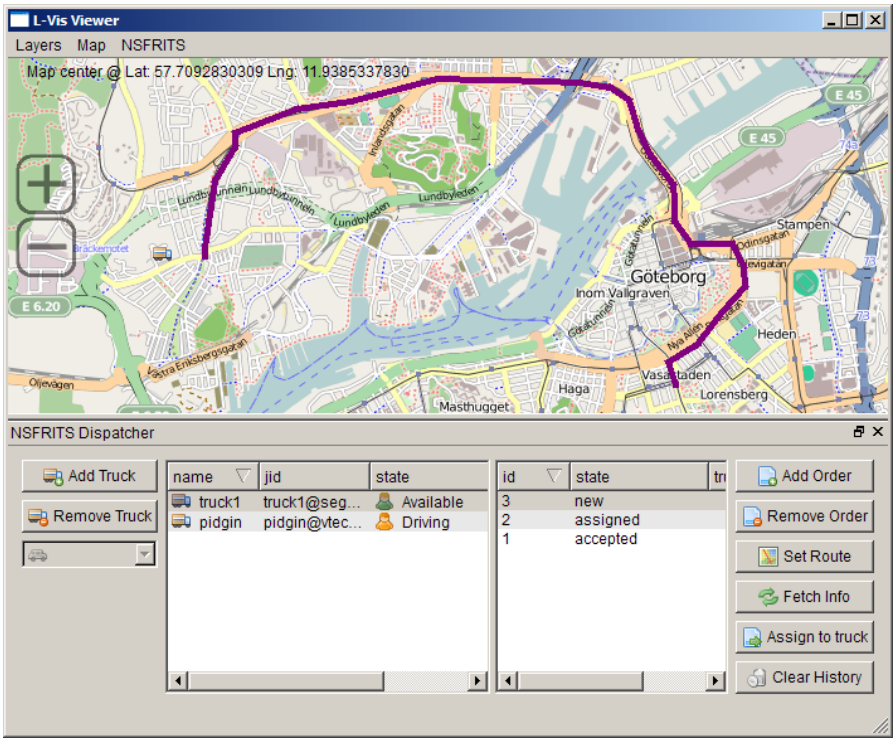

Figure 5.5.: Main window of the dispatcher client. Here the operator can see the status of trucks and orders, and the trucks current location.

plications, the NS-FRITS server, the NS-FRITS database interface, the Administrator interface and the DATEX II parser.

5.4.1. Overview

All NS-FRITS backend applications are written in Java and are interconnected with each other as shown in figure 5.6. The NS-FRITS database interface is the component responsible of storing and loading data from the database. The NS-FRITS core server provides access to NS-FRITS Clients through XMPP. The Administrator interface lets operators modify the data objects stored inside NS-FRITS. The DATEX II parser loads DATEX II XML files, converts them into NS-FRITS format, then inserts them into the NS-FRITS database.

5.4.2. NS-FRITS database interface

The NS-FRITS database interface is the module that provides objects to access the backend NS-FRITS database. It has been implemented through the design pattern Data

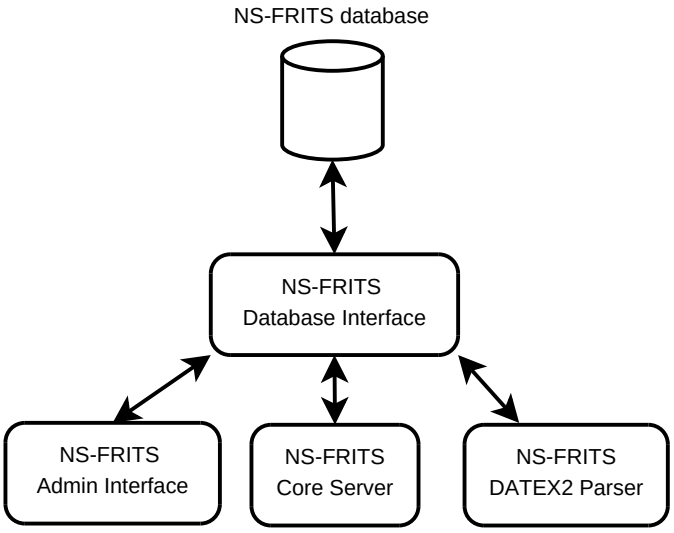

Figure 5.6.: Overview how the components in the NS-FRITS backend interrelate with each other

Access Object (DAO).

Data Access Object

A DAO is an object that provides certain functions to modify the underlying database. Three different DAOs has been created: ZoneDAO, ServiceDAO and UsersDAO. Each of these DAO provides CRUD (Create, Read, Update and Delete) functions for different underlying database tables. The ServiceDAO is responsible for categories and providers, the UsersDAO for users in the NS-FRITS system and the ZoneDAO for location based data such as infoobjects, descriptions and datanodes. Each of these objects also performs ORM (Object Relational Mapping), that means that the relational data in the database is mapped into Java objects when retrieved through the DAO functions. For example if the ZoneDAO function getInfoObject(id) is called, which retrieves the infoobject with the argument id, the result is a java object of the type "InfoObject".

Converting objects into other formats

Each object representing data in the NS-FRITS data model, such as InfoObject, Description, Category and DataNode, have been extended with a toXML function. The toXML function converts the object into an XML string, which is used by the NS-FRITS core server before sending the object to the NS-FRITS user. Some objects also have a toJSON function implemented, which is used to convert the object into JSON-format which is used by the administrator interface (See more in 5.4.4).

Database

The backend database selected in the implementation was SQLite. The main reason for this selection was that it can very easily be moved between different systems, as the database consists of a single file and no external services are needed.

5.4.3. NS-FRITS Server

The NS-FRITS server is the application responsible of connecting to the XMPP server and providing an API to NS-FRITS clients. The library selected for XMPP communication was the Smack API[9]. The methods described in Section 4.4 is implemented by writing custom packet listeners. Since the incoming data is in XML format, the data is first parsed into Smack Java objects, which are then handled by the appropriate custom packet listener. The custom packet listeners then call the right DAO method from the NS-FRITS database interface and in case the method has any results, it is converted into XML and inserted into the response packet.

5.4.4. Administrator interface

The administrator interface is an application that connects to the NS-FRITS database through a Java web application, the implementation is based on the three tiered architecture. The technologies used for implementation is a mixture of Java servlets, Java Server Pages (JSP) and Asynchronous Javascript and XML (AJAX). The web application is then deployed on Apache Tomcat.

Functionality

There are two main functionalities in the administrator interface, editing providers in NS-FRITS (Figure 5.7) and editing locations and InfoObjects bound to a certain provider (Figure 5.8). The edit provider part simply provides a CRUD interface for all data providers in NS-FRITS. It is possible to set what category each provider belongs to, change their name etc. For each provider in NS-FRITS it is also possible to create locations on a map and bind an InfoObject to them. The administrator can also modify each InfoObject by creating a tree of datanodes (See section 4.6.2), adding descriptions and uploading files.

Libraries

The library used for drawing maps is OpenLayers. OpenLayers is a Javascript library with support for fetching map tiles from several major map providers and draw overlays

[9] http://www.igniterealtime.org/projects/smack/

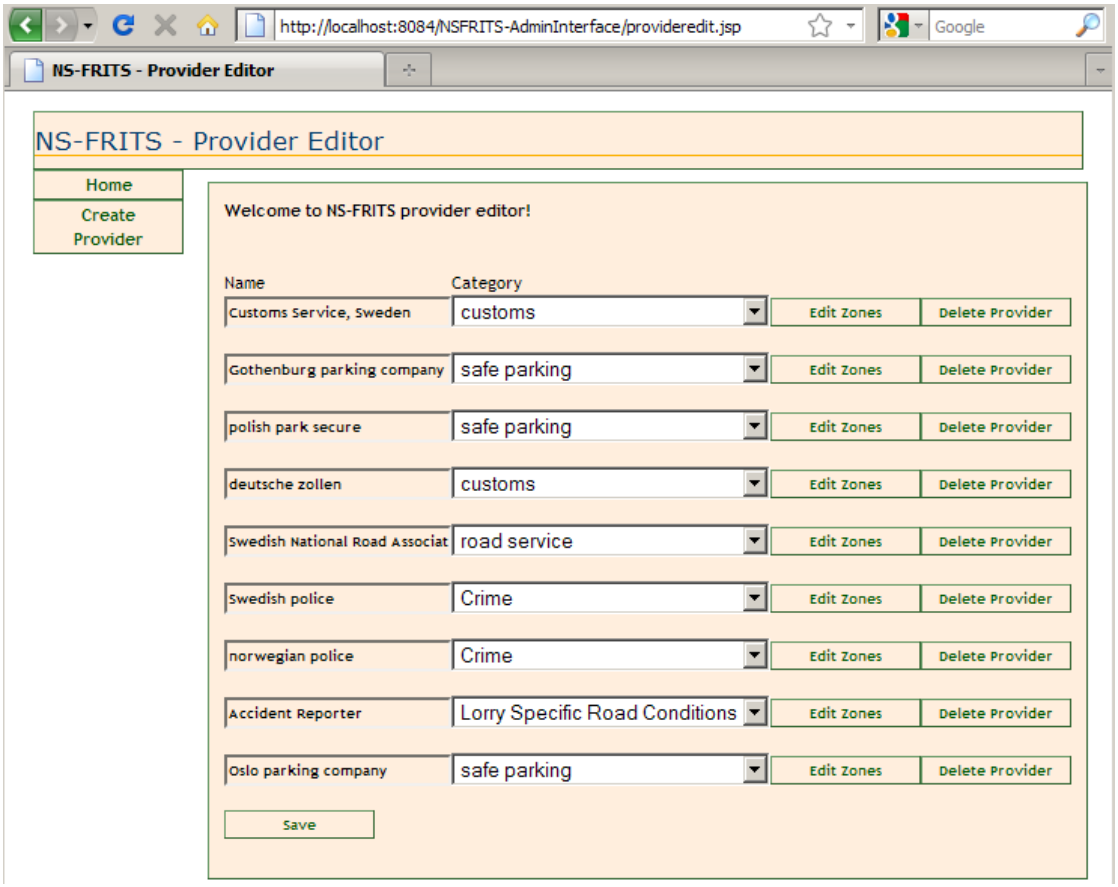

Figure 5.7.: Screenshot of the the editing providers view of the administrator interface. By clicking on the "Edit Zones" button, an administrator can edit the locations of each provider.

on them. The map provider chosen for this project was OpenStreetMaps(OSM) (more about map providers and projection in section 3.5). For AJAX calls the Javascript framework, Prototype[10], was selected and for file upload, the script Uploadify[11].

5.4.5. DATEX II parser

The last NS-FRITS backend application is the DATEX2 parser. This program simply loads DATEX II XML-files, parse the data and then converts it into the NS-FRITS format. This data is then stored into the NS-FRITS data model. For demonstrating this application, a couple of XML DATEX II files from the Swedish National Road Administration were gathered. A special data provider called "Swedish National Road

[10] http://www.prototypejs.org/
[11] http://www.uploadify.com/

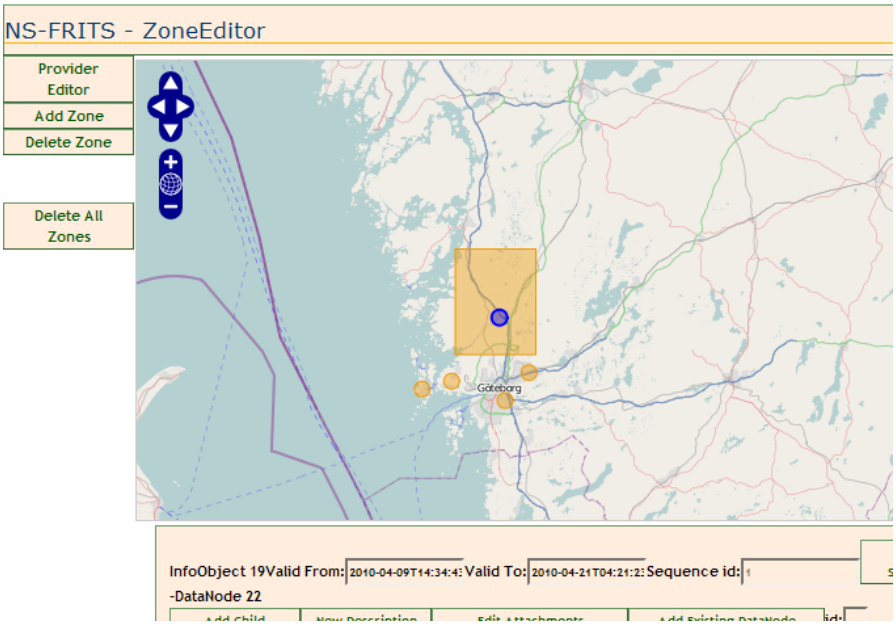

Figure 5.8.: Screenshot of an administrator editing the locations of a particular provider. The box in the middle represents the area where the location is active.

Administration" was then created through the administrator interface. The application were then set to parse these XML-files and then insert them into the NS-FRITS database for this particular data provider.

5.5. Time agreement application

The time agreement application was developed to showcase an example of a public NS-FRITS extension described in section 4.1.3. This application is also an example of an implementation of one of the pre-defined protocols mentioned in section 4.1.4. The scenario used for implementation was an application for the customs station in Svinesund. Trucks would be able to register their arrival time to the customs station. The customs personnel at the station would in their turn be able to see registered trucks through a graphical interface and inform them directly of delays at the station. If a truck notice that it cannot keep its registered time, it can send an update to the customs station containing a new arrival time.

Like the NS-FRITS backend applications, the time agreement application is also implemented in Java. It consists of two different parts, a Java web application for customs personnel to see registered trucks and a regular Java application to connect the application to the NS-FRITS XMPP server. The regular Java application will then act as a server storing registrations in a local database, the same database is then read by

the Java web application and displayed to customs personnel. A screenshot of the web application can be seen in Figure 5.9.

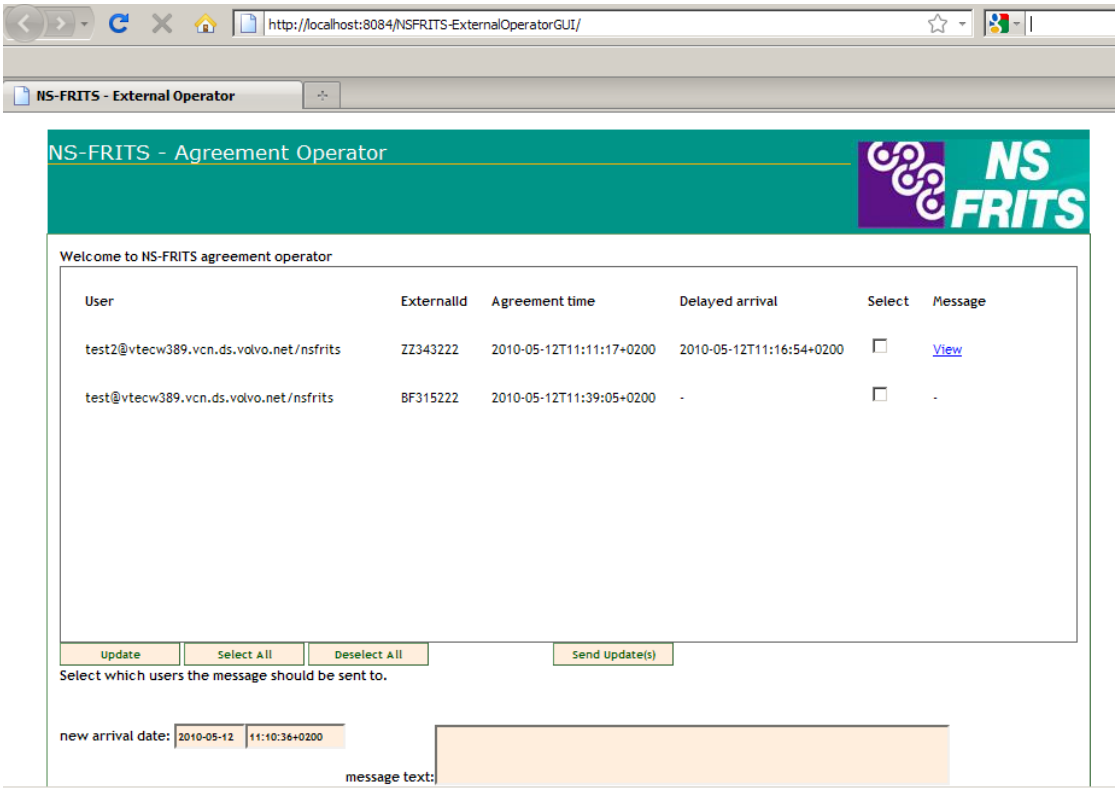

Figure 5.9.: Screenshot from the operators view of the implemented time agreement application.

Since the customs time agreement application is as a public NS-FRITS extension(see section 4.1.3), a link to its address is stored inside the NS-FRITS system. Whenever a truck gets information about the customs station in Svinesund, for example by planning a route through Svinesund, it will get the information that the Svinesund customs station also contains a time agreement application. An example of a user registering a time arrival can be seen in Figure 5.10.

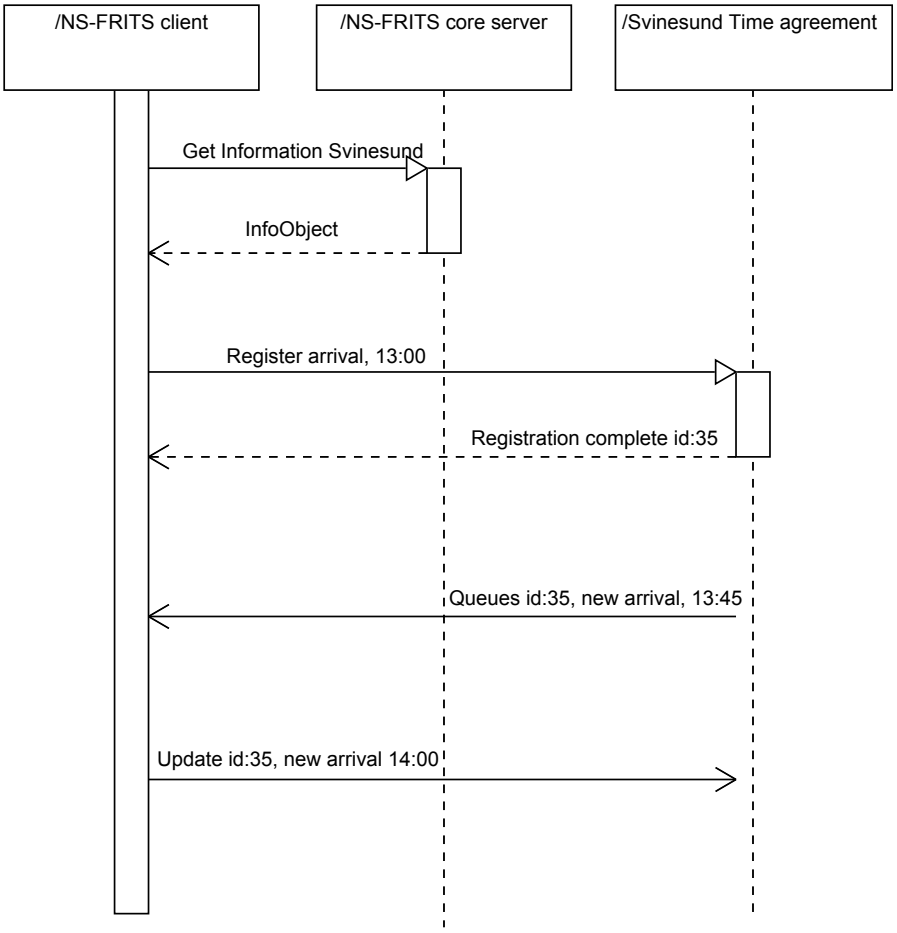

Figure 5.10.: Sequence diagram of a user registering an arrival time to the customs station. The customs station can in its turn send new arrival time to the user because of for example queues. The client will lastly send an updated time of his arrival.

6. Improvements

Since the goal of this project only was to design a prototype of the upcoming NS-FRITS system, there were several topics that could not be included in the project design because of the limited time frame. Some of these topics will be discussed in this chapter, which will be devoted to improvements and extensions of the NS-FRITS systems. These are some of the topics that might be interesting in future work and when the real NS-FRITS system is to be developed. This section will not only discuss what can be done in the future, but also give examples of how it could be implemented and extended into the current NS-FRITS prototype developed by this project.

6.1. Route planning

One topic that was discussed from the beginning of this project but not implemented in the NS-FRITS prototype was route planning. The idea is that a user would send a start, a destination and a set of parameters to the NS-FRITS system. Parameters can for example be, avoid crime zones if possible, visit a certain set of POIs in a certain order, avoid routes with certain road conditions such as construction work or optimize the route depending on variable road speed limits. The NS-FRITS system will then plan a route matching these parameters and send it back to the client.

6.1.1. Basic principles of route planning

Although route planning can be a very difficult and complicated task, there are some basic principles that applies to most route planning algorithms. Usually the roads are modeled as weighted edges in a graph. The weight of the edges can either be the length of the road or a product of the road length and the allowed driving speed of the road. The user gives a starting point and a destination point to the algorithm, then the fastest route is found via a shortest path algorithm such as Dijkstras[6] or A*. The user might not know the exact coordinates of its destination, so there need to be an intermediate layer transforming locations described in strings, such as “Street x, City y”, to geographical coordinates, this process is called geocoding. After transformation, the only task left is to find the closest road to the coordinate.

Even if the algorithm used has a big impact on the quality of the route planner, the biggest and most crucial step for an efficient route planner is the road map used. There are many

parameters that need to be stored about each road to be able to build an efficient graph, for example: The size of the road(i.e no Heavy Goods Vehicles), speed limit, traveling directions, turning restrictions, no through traffic, etc. All these parameters can have a major effect on the route chosen.

6.1.2. Route planning with NS-FRITS data

When NS-FRITS data is considered in the planning, the process gets more complicated. The NS-FRITS data model need to be extended to support these various plans, for example a crime hot spot need to be stored so the route planner understands that it should avoid this route. The simplest way to deal with the problem mentioned above is to simply increase the weight of the edges passing through crime hot spots, so the route planner is less likely to pick them.

One of the other premises that would be considered when planning routes is traffic incidents, such as construction works, road accidents or bad road weather. By parsing certain DATEX II tags, it is possible to calculate a new weight of an edge. For example, the `<overallImpact>` tag contains information about the severity of the event and the `<situtionRecord>` tag contains a time interval when the event is expected to occur and the type of the event, like for example ConstructionWorks, AbnormalTraffic or WeatherRelatedRoadConditions[10].

By then taking all these tags into consideration when NS-FRITS receives a DATEX II update, it is possible to calculate a new weighted value and apply it to the edge at the location described in the DATEX II message. This new weight can be applied temporarily until the expire date of the situation has passed. However, to find a good weighting value that can be used in the real world, one need to analyze how the DATEXII standard are implemented today by road administration services in different European countries.

6.1.3. Integration with NS-FRITS prototype design

Route planning would be easy to integrate with the current NS-FRITS prototype, the only change that would be needed to the server is an update of the API to include the route planning method. Since route planning can be computationally heavy it might be a good idea to deploy the route planning algorithm it self on a separate machine.

When route planning with NS-FRITS data is considered, it can be accomplished in different ways: Either one can change the NS-FRITS data model to include how each event affects the weight of the route planning algorithm, or one could keep the old data model unchanged and instead, in parallel, feed new NS-FRITS objects directly into route planner graph. An illustration of the first method can be seen in Figure 6.2, and the second method in Figure 6.1. The obvious advantage with the second method is that is can be built on top of the original NS-FRITS prototype without any changes, however, one will then need to maintain two different data structures, which leads to

higher system complexity. Another advantage with the first method is that the route planning functionality are not limited to the NS-FRITS system. If NS-FRITS stores event in a more formal way, third party developers can build their own route planner service by using data from the NS-FRITS system (more about how this can be achieved in 6.1.5.

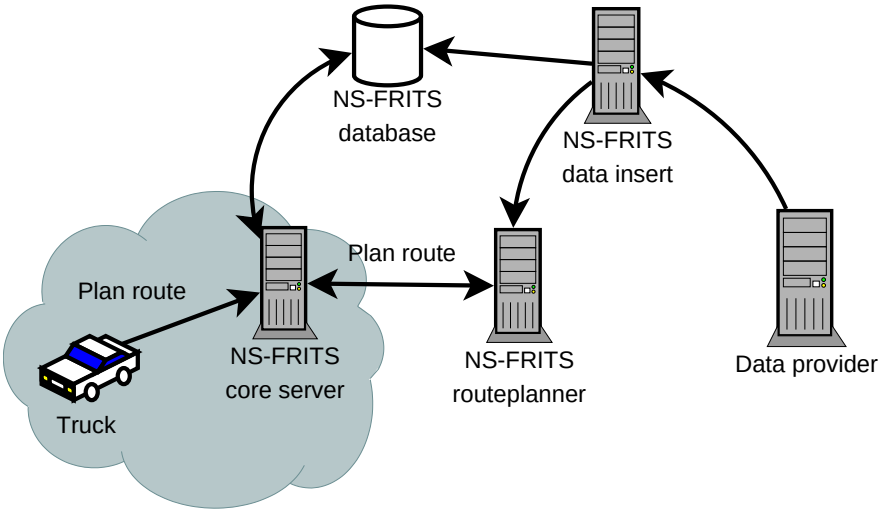

Figure 6.1.: Illustrates how route planning would be implemented without changing the original NS-FRITS system. When a data providers inserts new data, it is both inserted into the NS-FRITS database and the NS-FRITS route planner's data model.

6.1.4. Example Implementation with OpenStreetMaps and pgRouting

To demonstrate route planning, a simple route planner was implemented for the NS-FRITS prototype. This was just implemented as a proof-of-concept, and should not be considered to be part of the NS-FRITS design. Although some of its principles could be used for an implementation of a real route planner.

The example implementation uses road maps from OpenStreetMaps, storage in an PostgreSQL database[1] with PostGIS support[2] and routing algorithms from the PostLBS project pgRouting[3]. The pgRouting project consists of a set of functions implementing route planning functionality directly at database level[41]. Several searching algorithms are implemented, like Dijkstras algorithm, A* and a heuristic approximation to the traveling salesperson problem.

[1] `http://www.postgresql.org/`
[2] `http://postgis.refractions.net/`
[3] `http://pgrouting.postlbs.org/`

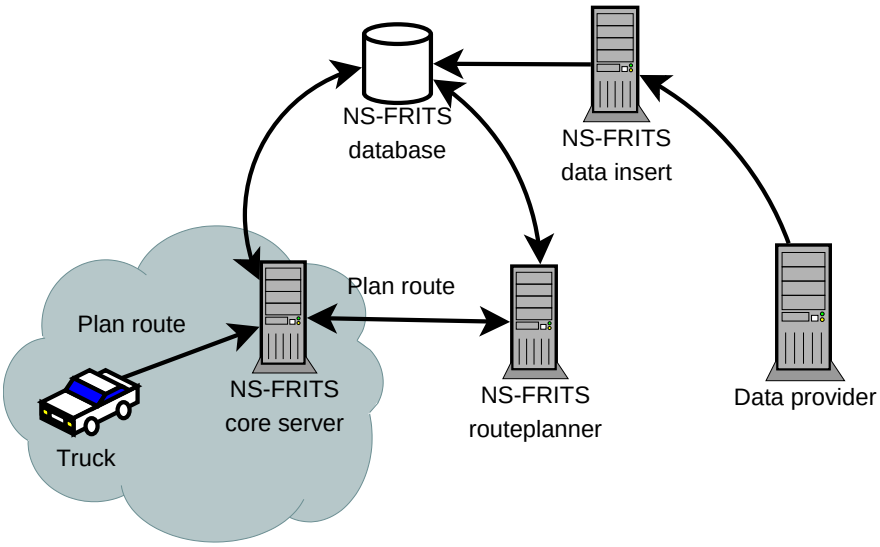

Figure 6.2.: Illustrates how route planning would be implemented by changing the NS-FRITS data model. In this method all data is inserted into the NS-FRITS database. The route planning server updates its internal data structure by querying the NS-FRITS database.

Implementation process

First, road maps was downloaded from OpenStreetMaps. The maps are downloaded as large XML-files, each containing data for different parts of the world. The downloaded maps are then inserted into the PostgreSQL/PostGIS database with the osm2pgrouting script. This script converts OpenStreetMaps data to a graph network readable by the pgRouting algorithms. When the data is inserted in the database, the pgRouting searching algorithms are run as SQL functions, returning the set of nodes visited for that particular route. Finally, the project wrote a wrapper to these functions in the java server, which queried the pgRouting algorithms and converted the result into a WKT Linestring. The routeplanning method was then added to the NS-FRITS API, taking a starting and a destination point as arguments and returning a WKT Linestring of the planned route. A screenshot representing the clients view of the route planning implementation can be seen in figure 6.3.

6.1.5. Third party route planning

Rather than letting NS-FRITS plan the route, another option is to just present data in a way so third party developers can utilize NS-FRITS in their own route planners. In the current NS-FRITS prototype this is possible to some extent by using existing NS-FRITS methods like searchPOI, getInfoForPoint and getInfoForRoute to get information about

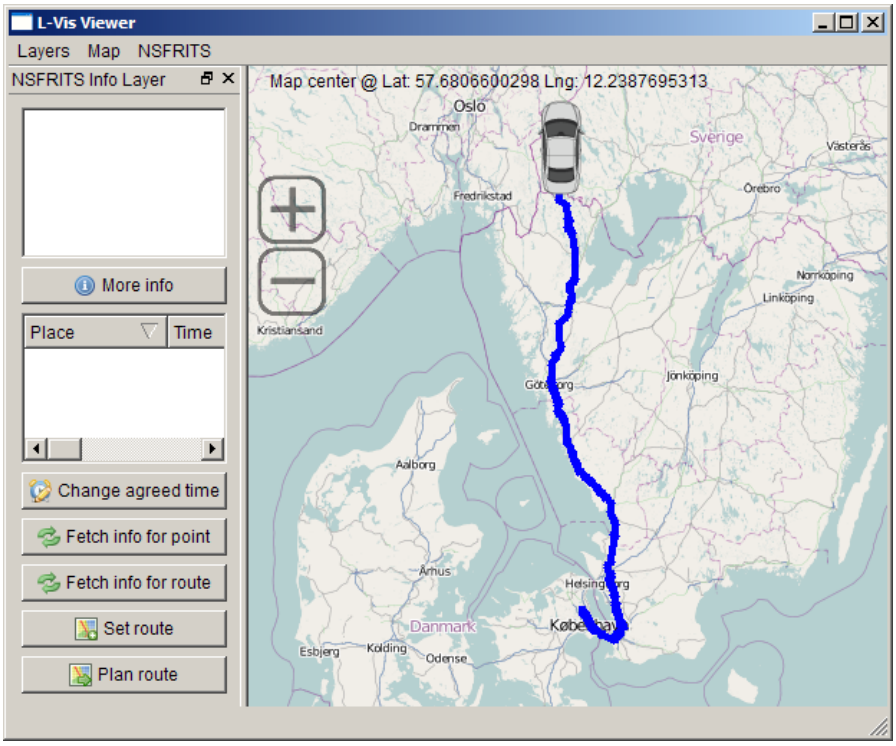

Figure 6.3.: Screenshot of the client using the example route planning implementation. The client sets a start and a destination point and then sends it to the server. The server calculates a route and then responds with a linestring, which the client draws out on the map.

specific routes. In the future the NS-FRITS data model might be extended to store more extensive information in each InfoObject to allow more advanced planning. With that in mind, the real bottleneck of being able to provide detailed data for third party route planners is not inside NS-FRITS, it is instead the quality and level of detail of the provided data from data providers.

6.2. Spatial algorithms

When the final NS-FRITS system is to be deployed, it is expected to work well with thousands of users and information objects. One of the areas that could be improved is the spatial algorithms used for finding out which information objects to send a user when he/she is planning a route. The geometrical object intersection algorithm used today's complexity scales with the number of edges in an object, so the obvious first measure to provide higher scalability is to simply limit how many edges an object can have. The computations needed at the client side can be reduced by more extensive use of the place

data type, which was described in section 4.6.3. Instead of sending a large polygon with a large amount of edges to represent, for example, a country or a city, a string can be used to describe the location. The client can then find out where it is located by using the local GSM network.

6.3. Further extensions

6.3.1. SOAP

The current design uses a custom protocol for method calls and responses. In the final NS-FRITS system, one might want to use a standardized protocol instead, like SOAP. The XMPP standards foundation has defined an extension for SOAP communication over XMPP in extension XEP-0072[19].

6.3.2. Translation and text-to-speech

Another possible area of improvement is translation and text-to-speech. In section 3.7.3, different translation principles were mentioned, a future project might want to refine these suggestions and design an infrastructure for them. Future projects might also want to look more deeply into text-to-speech. Text-to-speech can be implemented in either server side or client side. In the first option, the server uses a text-to-speech algorithm or library, converts the text into an audio file and then inserts a link to the file in the soundURL field of the description (see section 4.6.4). In the second option, the client implementation of NS-FRITS instead runs the text-to-speech software, thus the audio is never sent over the network. There are several text-to-speech libraries available today: Eyes-free for the Android platform [4], Natural Voices from AT&T [5] or Festival [6] which is an open source project.

6.3.3. NS-FRITS data converter module

In section 4.1.2, a data converter module was mentioned for insertion of data in a non-NS-FRITS format, for example DATEX II. The data converter would either subscribe to an existing feed of information or periodically fetch the data itself, the data would then be converted and inserted into NS-FRITS. As a future project, one might want to implement one of these data converter modules, for example for DATEX II data. Since a parser for DATEX II already has been implemented (see section 5.4.5), the purpose of this module would just be to connect to existing DATEX II providers around europe and insert data into NS-FRITS whenever an update occurs.

[4] `http://code.google.com/p/eyes-free/`
[5] `http://www2.research.att.com/~ttsweb/tts/demo.php`
[6] `http://www.cstr.ed.ac.uk/projects/festival/`

7. Results

The following results was presented at the end of this thesis work:

- A proposed system design for the NS-FRITS system was presented.
- The proposed design was implemented into a working prototype.
- Improvements for further development of the NS-FRITS system was suggested.

At the end, all goals set in section 1.5 were reached. The achieved results of this project was presented at the NS-FRITS project conference in Bremerhaven, Germany in June 2010.

8. Discussion

The outcome of this project can be considered a success, since the goals set in the start of the project has been reached and a working prototype of the system has been developed. The development process has worked out according to schedule with no major changes. The only problem the project had was that the system requirements changed several times during the design and implementation stages. However, since an iterative development methodology was chosen in the start of the project, this was to be expected.

We believe that the developed prototype design for the NS-FRITS system can both be used for the final NS-FRITS system which will be completed in late 2011 and for future ITS project within the area of location based services.

8.1. Technology evaluation

The XMPP protocol, which was used for communication was a good choice for data transfer since it brings always-on-functionality to roaming trucks and clients. XMPP seems to be a good choice since it provides a lot of extensibility and scalability by using multiple servers, see section 4.2. In the future, when IPv6 has been more widely deployed, the need of an intermediate communications layer such as XMPP might not be needed anymore. The communication can then be handled directly by host-to-host techniques like MobileIPv6. However, this does not affect the NS-FRITS system developed in this thesis work since the same data protocol and server mechanisms can be used, the only part that needs to be changed is the communications layer.

8.2. Future work

In chapter 6, the thesis also discussed how the design and implementation of the NS-FRITS prototype could be extended and improved for future work. One major area was the topic of automatic route planning. An interesting future project would be to design and implement an advanced route planner that considers NS-FRITS data while planning. Another interesting project would be to investigate more rigorously how NS-FRITS data could be used to improve existing route planners in the market.

In section 4.1.3, the concept of a NS-FRITS application was introduced, an example of this kind of application was introduced with the time agreement protocol. A future

project could consist of designing and implementing more NS-FRITS applications. For example for booking services or electronic document transfer and signing.

Lastly, the prototype developed need to be tested and evaluated in a real environment with real data providers.

Bibliography

[1] Apple Developer: Safari Dev Center. http://developer.apple.com/safari/, April 2010.

[2] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Tim Schaub, and Christopher Schmidt. The GeoJSON Format Specification. http://geojson.org/geojson-spec.html (accessed 2010-05-06), June 2008.

[3] Ian Channing. Twelve operators commit to LTE deployment in 2010. http://www.fiercewireless.com/europe/story/twelve-operators-commit-lte-deployment-2010/2009-06-17, June 2009.

[4] Sanphet Chunithipaisan and Soravis Supavetch. The development of web processing service using the power of spatial database. *Emerging Trends in Engineering & Technology, International Conference on*, 0:832–837, 2009.

[5] European Commission. Intelligent Transport Systems and Services: initiative for accelerated deployment across Europe IP/08/1979. http://europa.eu/, December 2006.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* The MIT Press, 2nd edition, September 2001.

[7] Dave Crane and Phil McCarthy. *Comet and Reverse Ajax: The Next-Generation Ajax 2.0.* Apress, Berkely, CA, USA, 2008.

[8] DATEX Background. http://www.datex2.eu/content/datex-background, 2009.

[9] DATEX II V2.0 Software Developers Guide. http://www.datex2.eu/sites/www.datex2.eu/files/sites/test.datex2.eu/files/DATEXIIv2.0-DevGuide_v1.0.pdf, July 2009.

[10] DATEX II V 2.0 User Guide. http://www.datex2.eu/sites/www.datex2.eu/files/sites/test.datex2.eu/files/DATEXIIv2.0-UserGuide_v1.0.pdf, July 2009.

[11] Michael N. DeMers. *GIS For Dummies.* Wiley Publishing, Inc., Hoboken, NJ, USA, 2009.

[12] OpenLayers Developers. Open Layers Library Documentation. http://docs.openlayers.org/library/spherical_mercator.html.

[13] Jean Dollimore, Tim Kindberg, and George Coulouris. *Distributed Systems: Concepts and Design (International Computer Science Series)*. Addison Wesley, 4th edition edition, May 2005.

[14] Ahmed El-Rabbany. *Introduction to GPS: The Global Positioning System*. Artech House, Sussex Street, London, 2. ed. edition, 2006.

[15] Roaming: High Prices of SMS & Data Services . `http://europa.eu/rapid/pressReleasesAction.do?reference=MEMO/08/505`, July 2008.

[16] Knut Evensen. IEEE 802 CALM Tutorial. `http://grouper.ieee.org/groups/802/11/Reports/Nov_2006_CALM_Tutorial/IEEE_802_CALM_Tutorials_PPT.zip`, 2006.

[17] Stephen Ezell. Explaining International IT Application Leaderhip: Intelligent Transportation Systems. Technical report, ITIF, January 2010.

[18] International Organization for Standardization. Codes for the Representation of Names of Languages Part 2: Alpha-3 Code. `http://www.loc.gov/standards/iso639-2/php/code_list.php`, 2002.

[19] Fabio Forno and Peter Saint-Andre. XEP-0072: SOAP Over XMPP. Technical report, XMPP Standards Foundation, December 2005.

[20] Google Chrome: FAQ for web developers. `http://www.google.com/chrome/intl/en/webmasters-faq.html`, April 2010.

[21] Lawrence Harte. *Introduction to Wireless Local Area Network (WLAN) technology, market, operation, profiles and services*. ALTHOS Publishing Inc, Fuquay Varina, N.C., 1. ed. edition, 2004.

[22] Michi Henning. The rise and fall of CORBA. *Commun. ACM*, 51(8):52–57, 2008.

[23] Ailing Huang, Jinsheng Shen, and Wei Guan. ITS planning methodology for Chinese cities and its evaluation model. In *Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE*, pages 1125 –1130, 2006.

[24] Open Geospatial Consortium Inc. *OpenGIS Geography Markup Language (GML) Implementation Specification*, 3.1.1 edition, February 2004.

[25] Open Geospatial Consortium Inc. *OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture*, 1.2 edition, October 2006.

[26] Open Geospatial Consortium Inc. *OGC KML*, 2.2.0 edition, April 2008.

[27] ISO TC 204 WG 16. What is Working Group 16 CALM Concept. `http://www.isotc204wg16.org/concept`, 2008.

[28] Bonghyun Jeong. Institutional issues to successful ITS implementation in Korea. In

Vehicle Navigation and Information Systems Conference, 1996. VNIS '96, volume 7, pages 118 – 125, 1996.

[29] Daniel Jiang and Luca Delgrossi. IEEE 802.11p: Towards an international standard for wireless access in vehicular environments. In Mario Gerla and Yuming Jiang, editors, *VTC2008-Spring, 67th IEEE Vehicular Technology Conference*, pages 2036–2040, Los Alamitos, CA, USA, May 2008. IEEE Computer Society.

[30] D. Elliott Kaplan and Christopher J. Hegarty. *Performance of Stand-Alone GPS*. Artech House, Sussex Street, London, 2. ed. edition, 2006.

[31] Martin Karlsson and Edvin Valtersson. *L-ViS - LDM Visualization System*. Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden, 2009.

[32] Farooq Khan. *LTE for 4G Mobile Broadband: Air Interface Technologies and Performance*. Cambridge University Press, Brook Hill Drive West Nyack, NY, 1. ed. edition, 2009.

[33] Juha Korhonen. *Introduction to 3G mobile communications*. Artech House, Sussex Street, London, 2. ed. edition, 2003.

[34] North Sea Freight and Intelligent Transport Solutiuons (EU project 35-2-38-08). Scenario - actors and use cases. (work in progress), February 2010.

[35] North Sea Freight and Intelligent Transport Solutiuons (EU project 35-2-38-08). System design. (work in progress), March 2010.

[36] North Sea Freight and Intelligent Transport Solutiuons (EU project 35-2-38-08). System requirements. (work in progress), March 2010.

[37] NS FRITS Aims and Objectives. `http://www.nsfrits.eu/en/ns-frits/aims-and-objectives.html`, 2009.

[38] IEEE P802.11 Task Group p. Status of Project IEEE 802.11 Task Group p, Wireless Access in Vehicular Environments (WAVE). `http://grouper.ieee.org/groups/802/11/Reports/tgp_update.htm`, 2010.

[39] S. Parkvall, E. Dahlman, A. Furuskar, Y. Jading, M. Olsson, S. Wanstedt, and K. Zangi. LTE-Advanced - Evolving LTE towards IMT-Advanced. In *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, pages 1–5, Sept. 2008.

[40] Eldad Perahia and Robert Stacey. *Next Generation Wireless LANs: Throughput, Robustness, and Reliability in 802.11n*. Cambridge University Press, Brook Hill Drive West Nyack, NY, 1. ed. edition, 2008.

[41] pgRouting Documentation . `http://pgrouting.postlbs.org/wiki/pgRoutingDocs/`, June 2009.

[42] Peter Saint-Andre. Extensible messaging and presence protocol (XMPP): core. RFC 3920, IETF, October 2004.

[43] Peter Saint-Andre. Extensible messaging and presence protocol (XMPP): instant messaging and presence. RFC 3921, IETF, October 2004.

[44] Peter Saint-Andre. Streaming XML with Jabber/XMPP. *Internet Computing, IEEE*, 9(5):82 – 89, sept.-oct. 2005.

[45] Peter Saint-Andre, Kevin Smith, and Remko Tronçon. *XMPP: The Definitive Guide: Building Real-Time Applications with Jabber Technologies.* O'Reilly Media, Inc., Sebastopol, CA, USA, May 2009.

[46] Svensk författningssamling: Förordning om gränstullsamarbete med Norge, SFS 2002:1054. Stockholm, 2002.

[47] William Stallings. *Wireless Communications & Networks.* Prentice Hall, Upper Saddle River, NJ, 2. ed. edition, 2005.

[48] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems. Principles and Paradigms.* Prentice Hall International, 2nd rev. ed. edition, October 2006.

[49] Volvo Technology webpage. `http://www.tech.volvo.com/`, February 2010.

[50] Dave Winer. XML-RPC Specification. `http://www.xmlrpc.com/spec#update3/`, June 2003.

[51] GSM World. Market Data Summary (Q2 2009). `http://www.gsmworld.com/newsroom/market-data/market_data_summary.htm`, October 2009.

A. Use case

A.1. Driver A traveling Warsaw to Oslo via Svinesund

- Driver A is scheduled to run goods from Warsaw, Poland to Oslo, Norway. This is the first time driver A is driving goods to Norway, so the driver is not familiar with the customs procedure when leaving EU to enter Norway.
- When driver A is 50 km away from the Sweden-Norway border at Svinesund the in-vehicle device alerts the driver. A question is asked by a voice in the native language of the driver, the same text is displayed on the screen: You are approaching the Svinesund customs station, would you like to know more about the procedure at this station? (Yes/No)
- Since the driver never has traveled through this border before the driver selects yes. A voice starts to speak in polish about how the customs are handled there, and where to go for goods declaration. A map of the border station is also shown on the screen so the driver can see key points where to go. The driver can anytime via the in-vehicle system read more or listen to more detailed description if he wishes to know more about the customs station and the procedures there.
- Driver A approaches the station, since he already received information about where to go for goods declaration and what papers he need to sign he does not need to ask for help at the station and his passing through customs will be as effective as possible.

A.2. Driver A traveling Oslo to Warsaw via Svinesund

- Driver A has reached his goal in Oslo and delivered his goods. The driver will now return to Warsaw without any cargo.
- When driver A is 50km away from the Norway/Sweden border the driver gets info on what to do when crossing the border with an empty truck. He is not required to stop at the customs station, instead can drive though without stopping.
- Driver A can drive past the station without having to stop to ask what to do.

B. Case study: The Swedish Customs' office in Svinesund

The following is a case study of the Swedish customs' office in Svinesund at the Norwegian border. The information is, unless specified, from interviews with customs employees and visits to the customs office.

B.1. Description

The border between Sweden and Norway is interesting not only because of being an outer EU border, but also for the unique treaty that allows the both countries' customs agencies to collaborate very closely at and around the border[46]. This treaty allows a customs officer to take the role of both parties and do both the import and export declarations at the same time. This is important since the border is very long and, except for a few places, very sparsely trafficked and by dividing the customs offices between them a lot of money and resouces can be saved. Only at the larger border crossings in Svinesund and Hån/Örje are officers from both countries present.

At the south most point on the border between Sweden and Norway lies the small community of Svinesund. Here the road E6 leave the EU and enter Norway. This road is one of the main roads for cargo transports to and from Norway and the customs office gets visited by drivers from all over Europe.

The traffic from Sweden to Norway is not stopped on the Swedish side as customary, instead the traffic is allowed over the border into Norway where the Norwegian customs handles both the export from the EU and the import into Norway. For traffic traveling in the other direction it is the opposite, here the Swedish customs do both the export from Norway and import into EU.

B.2. Problem

- Customs inspection is located after the border crossing for transports in both directions. This is unique for the Swedish-Norwegian border. See figure B.1.
- A toll plaza for the border bridge is located close to the customs building on the Swedish side. Toll in Norwegian means customs.

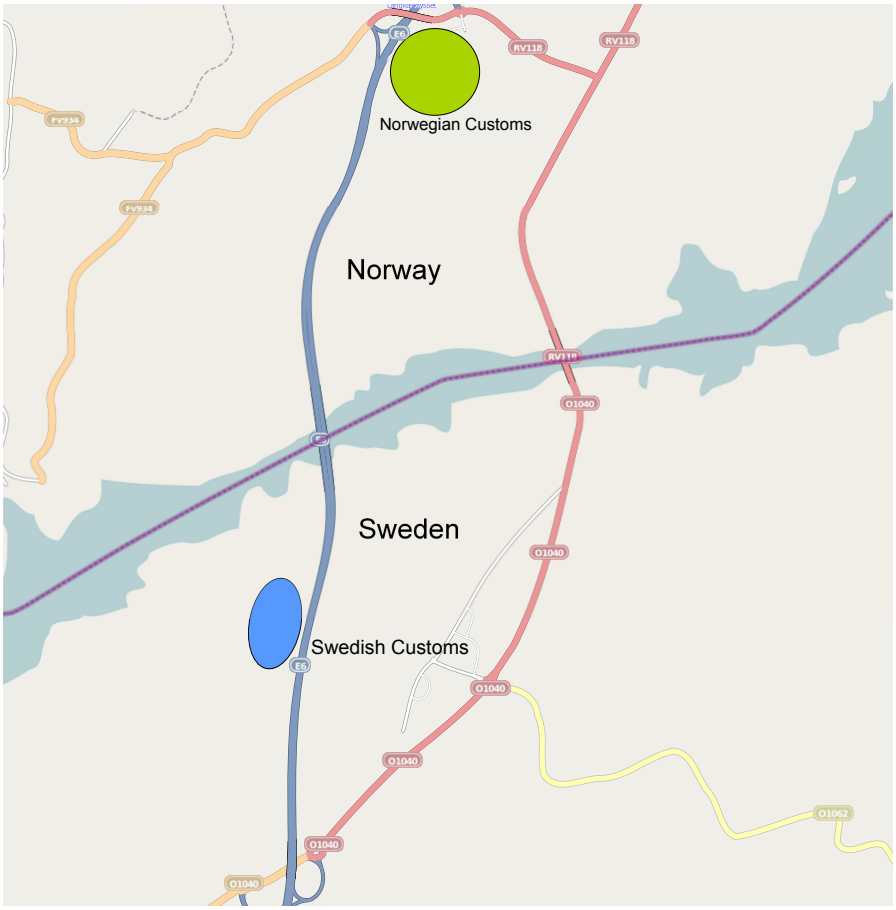

Figure B.1.: Map view of the Svinesund area. The circles represents the customs offices, where the Norwegian handles the northbound traffic and the Swedish office handles the southbound traffic. ©OpenStreetMap & contributors, CC-BY-SA

- No gates or armed guards to clearly show where to drive. Very low security compared to many places in especially eastern Europe, which makes it possible to miss the station if the driver expects a different form of station.
- Many drivers can not speak or understand English. Translating information to other languages is hard since all information has to be written and verified at a higher level in the organization.
- Empty transports does not have to stop for clearance, they can just drive by. Even though there are signs that informs about this, many drivers still stop.

B.3. Possible NS-FRITS Solution

The NS-FRITS system can improve the situation by helping the drivers to understand how the border passing works here and also help the customs officers to get important information out to the drivers quickly, all in a language the driver understands.

More specifically:

- The driver will be notified in advance that he is going to pass the border and have to have his documents ready and in order. The nofitication will be given so early so that the driver still has the option to stop at a appropriate location and fix his documents.
- The information point covering the border area will contain information about how to drive when passing the border in each direction.
- Maps over the area and other documents can be attached to the information point.
- General information will be automatically be translated and then proof read by NS-FRITS personel and by the agencies themselves.
- Important legal information will be translated into multiple languages by authorized translators to avoid confusion and misunderstandings.
- The customs agencies will get advance notice of arriving trucks that have announced that they will pass the border using the agreement service, and can thus plan the work load easier.
- Urgent updates can be sent to the trucks informing them about for example delays and accidents.

C. API

C.1. Location based information

Synchronous commands using the IQ stanza for communication with the NSFRITS location based information system.

Method:	getInfoForPoint
Argument:	point
Argument type:	WKT Point
Expected response:	list of infoobjects

Listing C.1: Example of a getInfoForPoint request and response.

```
<!-- Request -->
<iq id='qxmpp10' to='nsfrits@service.nsfrits.eu/nsfrits' type='get'>
  <query xmlns="nsfrits:iq">
    <methodcall>getInfoForPoint</methodcall>
    <arguments>
      <argument>
        <key>point</key>
        <value>POINT(11.878967 57.737883)</value>
      </argument>
    </arguments>
  </query>
</iq>

<!-- Response -->
<iq from='nsfrits@service.nsfrits.eu/nsfrits' to='truck1@service.nsfrits.
    eu/QXmpp' id='qxmpp10' type='result'>
<query xmlns='nsfrits:iq'>
<methodresponse>
<infoobject>
  <id>20</id>
  <validity>
    <from>2010-03-30 10:09:39</from>
    <to>2010-04-10 23:56:19</to>
  </validity>
  <sequenceid>1</sequenceid>
  <location>
    <area>POLYGON((11.546630859516 57.554946161479,11.546630859516
        57.825065399809,11.975097656374 57.825065399809,11.975097656374
        57.554946161479,11.546630859516 57.554946161479))</area>
    <point>POINT(11.644134521626 57.701266095542)</point>
  </location>
```

```
29    <datanode>
30      <id>23</id>
31      <description>
32        <title>Öckerö parking</title>
33        <text>Parking space for the Öckerö ferry.</text>
34        <language>en</language>
35      </description>
36      <attachment>
37        <mime-type>image/png</mime-type>
38        <size>389878</size>
39        <url>http%3A%2F%2F127.0.0.1%3A8084%2FServiceAdmin%2Fuploads%2F23%2
              Fcar.PNG</url>
40      </attachment>
41    </datanode>
42  </infoobject>
43  </methodresponse>
44  </query>
45  </iq>
```

Method:	getInfoForRoute
Argument:	-
Argument type:	-
Expected response:	list of infoobjects

Listing C.2: Example of a getInfoForRoute request and response.

```
1  <!-- Request -->
2  <iq id='qxmpp31' to='nsfrits@service.nsfrits.eu/nsfrits' type='get'>
3    <query xmlns="nsfrits:iq">
4      <methodcall>getInfoForRoute</methodcall>
5    </query>
6  </iq>
7
8  <!-- See getInfoForPoint for example response -->
```

Method:	getCategories
Argument:	-
Argument type:	-
Expected response:	list of categories

Listing C.3: Example of a getCategories request and response.

```
1  <!-- Request -->
2  <iq id='qxmpp34' to='nsfrits@service.nsfrits.eu/nsfrits' type='get'>
3    <query xmlns="nsfrits:iq">
4      <methodcall>getCategories</methodcall>
5    </query>
6  </iq>
7
8  <!-- Response -->
9  <iq from='nsfrits@service.nsfrits.eu/nsfrits' to='truck1@service.nsfrits.
       eu/QXmpp' id='qxmpp34' type='result'>
```

```
<query xmlns='nsfrits:iq'>
<methodresponse>
<category>
  <id>1</id>
  <description>
    <title>customs</title>
    <text>some text</text>
    <language>en</language>
  </description>
</category>
<category>
  <id>2</id>
  <description>
    <title>road service</title>
    <text>Road services</text>
    <language>en</language>
  </description>
</category>
<category>
  <id>4</id>
  <description>
    <title>Foreign Law</title>
    <text>Foreign laws deal with the fact that different countries, and other
regions like federal states or individual cities, may have different rules and r
egulations concerning truck transports. </text>
    <language>en</language>
  </description>
</category>
</methodresponse>
</query>
</iq>
```

Method:	setCategories
Argument:	categories
Argument type:	comma separated list of category id:s
Expected response:	-

Listing C.4: Example of a setCategories request and response.

```
<!-- Request -->
<iq id='qxmpp37' to='nsfrits@service.nsfrits.eu/nsfrits' type='set'>
  <query xmlns="nsfrits:iq">
    <methodcall>setCategories</methodcall>
    <arguments>
      <argument>
        <key>categories</key>
        <value>42,123,9</value>
      </argument>
    </arguments>
  </query>
</iq>
```

```
13
14 <!-- Response -->
15 <iq from='nsfrits@service.nsfrits.eu/nsfrits' to='truck1@service.nsfrits.
       eu/QXmpp' id='qxmpp37' type='result'>
16    <query xmlns='nsfrits:iq'/>
17 </iq>
```

Method:	setLanguages
Argument:	languages
Argument type:	comma separated list of language codes according to ISO 639-1
Expected response:	-

Listing C.5: Example of a setLanguages request and response.

```
1  <!-- Request -->
2  <iq id='qxmpp38' to='nsfrits@service.nsfrits.eu/nsfrits' type='set'>
3    <query xmlns="nsfrits:iq">
4      <methodcall>setLanguages</methodcall>
5      <arguments>
6        <argument>
7          <key>languages</key>
8          <value>sv,en</value>
9        </argument>
10     </arguments>
11   </query>
12 </iq>
13
14 <!-- Response -->
15 <iq from='nsfrits@service.nsfrits.eu/nsfrits' to='truck1@service.nsfrits.
       eu/QXmpp' id='qxmpp38' type='result'>
16    <query xmlns='nsfrits:iq'/>
17 </iq>
```

Method:	setRoute
Argument:	route
Argument type:	WKT Linestring
Expected response:	-

Listing C.6: Example of a setRoute request and response.

```
1  <!-- Request -->
2  <iq id='qxmpp41' to='nsfrits@service.nsfrits.eu/nsfrits' type='set'>
3    <query xmlns="nsfrits:iq">
4      <methodcall>setRoute</methodcall>
5      <arguments>
6        <argument>
7          <key>route</key>
8          <value>LINESTRING(11.948318 57.714785, 11.961365 57.724319,
               11.981277 57.726519, 11.992264 57.715518, 11.995010 57.696809,
                12.039642 57.617095, 12.065048 57.585455, 12.051315
               57.508447, 12.046509 57.477450, 12.075348 57.466374, 12.019043
                57.410941)
```

```
        </value>
      </argument>
    </arguments>
  </query>
</iq>

<!-- Response -->
<iq from='nsfrits@service.nsfrits.eu/nsfrits' to='truck1@service.nsfrits.
    eu/QXmpp' id='qxmpp41' type='result'>
   <query xmlns='nsfrits:iq'/>
</iq>
```

Method:	clearHistory
Argument:	-
Argument type:	-
Expected response:	-

Listing C.7: Example of a clearHistory request and response.

```
<!-- Request -->
<iq id='qxmpp44' to='nsfrits@service.nsfrits.eu/nsfrits' type='set'>
  <query xmlns="nsfrits:iq">
    <methodcall>clearHistory</methodcall>
  </query>
</iq>

<!-- Response -->
<iq from='nsfrits@service.nsfrits.eu/nsfrits' to='truck1@service.nsfrits.
    eu/QXmpp' id='qxmpp44' type='result'>
   <query xmlns='nsfrits:iq'/>
</iq>
```

Method:	searchPOI
Argument:	category, point, results
Argument type:	id of category(int), WKT point, number of maximum wanted results(int)
Expected response:	-

Listing C.8: Example of a searchPOI request and response.

```
<!-- Request -->
<iq id='qxmpp44' to='nsfrits@service.nsfrits.eu/nsfrits' type='set'>
  <query xmlns="nsfrits:iq">
    <methodcall>searchPOI</methodcall>
    <arguments>
      <argument>
        <key>category</key>
        <value>2</value>
      </argument>
      <argument>
        <key>point</key>
        <value>POINT(23.7 51.1)</value>
      </argument>
```

```
      <argument>
        <key>results</key>
        <value>3</value>
      </argument>
    </arguments>
  </query>
</iq>

<!-- Response -->
<iq from='nsfrits@service.nsfrits.eu/nsfrits' to='truck1@service.nsfrits.
    eu/QXmpp' id='qxmpp44' type='result'>
   <query xmlns='nsfrits:iq'>
    <methodresponse>
      <infoobject>
        <!-- content is left out -->
      </infoobject>
      <infoobject>
        <!-- content is left out -->
      </infoobject>
      <infoobject>
        <!-- content is left out -->
      </infoobject>
    </methodresponse>
   </query>
</iq>
```

Method:	addSubscription
Argument:	id
Argument type:	id of infoobject of that should be subscribed to
Expected response:	-

Listing C.9: Example of a addSubscription request and response.

```
<!-- Request -->
<iq id="lcRWE-4" to="nsfrits@service.nsfrits.eu/nsfrits" type="get">
 <query xmlns="nsfrits:iq">
  <methodcall>addSubscription</methodcall>
  <arguments>
    <argument>
      <key>id</key>
      <value>34</value>
    </argument>
  </arguments>
 </query>
</iq>

<!-- Response -->
<iq from='nsfrits@service.nsfrits.eu/nsfrits' to='truck1@service.nsfrits.
    eu/QXmpp' id='lcRWE-4' type='result'>
   <query xmlns='nsfrits:iq'/>
</iq>

```

```
<!-- This is send by the server whenever the subscribed infoobject is
    updated -->
<message from='nsfrits@service.nsfrits.eu/nsfrits' to='truck1@service.
    nsfrits.eu/QXmpp' id='U4QAq-4'>
  <updates xmlns='nsfrits:async'>
    <infoobject>
      <!-- content is left out -->
    </infoobject>
  </updates>
</message>
```

Method:	clearSubscriptions
Argument:	-
Argument type:	-
Expected response:	- removes all subscriptions for that user

Listing C.10: Example of a clearSubscriptions request and response.

```
<!-- Request -->
<iq id="lcRWE-4" to="nsfrits@service.nsfrits.eu/nsfrits" type="get">
 <query xmlns="nsfrits:iq">
  <methodcall>clearSubscriptions</methodcall>
 </query>
</iq>

<!-- Response -->
<iq from='nsfrits@service.nsfrits.eu/nsfrits' to='truck1@service.nsfrits.
    eu/QXmpp' id='lcRWE-4' type='result'>
   <query xmlns='nsfrits:iq'/>
</iq>
```

Method:	planRoute
Argument:	start,destination
Argument type:	WKT point
Expected response:	route containing a WKT linestring

Listing C.11: Example of a planRoute request and response.

```
<!-- Request -->
<iq id="lcRWE-4" to="nsfrits@service.nsfrits.eu/nsfrits" type="get">
 <query xmlns="nsfrits:iq">
  <methodcall>planRoute</methodcall>
  <arguments>
    <argument>
      <key>start</key>
        <value>POINT(18.009 59.304)</value>
    </argument>
    <argument>
      <key>destination</key>
      <value>POINT(12.493 55.795)</value>
    </argument>
  </arguments>
```

```
 </query>
</iq>

<!-- Response -->
<iq id="lcRWE-4" to="truck1@service.nsfrits.eu/nsfrits" type="result">
  <query xmlns="nsfrits:iq">
    <methodresponse>
      <route>LINESTRING (18.0075321 59.3037806, 18.0067597 59.3036404,
          18.006485 59.3032592, 18.0058842 59.3023186, 18.0056381
          59.3021355, 18.0051136 59.3017654, 18.0042667 59.3009262,
          12.4928965 55.7937747)
      </route>
    </methodresponse>
  </query>
</iq>
```

C.2. Time agreement

Registration of a new time agreement is done by synchronous IQ stanzas as shown in Listing C.12. The registration contains one mandatory field, agreetime, which contains the time of the agreement. Another two fields can optionally be used, jid and externalid. The jid field is used if the agreement is set up by a third party, like for example a fleet operator setting up an agreement for one of its trucks. The jid field should then contain the address of the truck. The external id field is used to link the agreement to an already existing agreement between the parties, like for example a booking number.

If the agreement is successful, the server will respond with an id of the created agreement. This id is later used to send and receive updates about the agreement.

Listing C.12: Example of a time agreement request.

```
<!-- Request -->
<iq id="qxmpp27" to="customsoperator@customsservice.domain/
    customsoperator" type="get">
  <query xmlns="operator:iq:register">
    <agreetime>2010-05-17T08:25:56+0200</agreetime>
    <jid>truck1@freightercompay.domain/QXmpp</jid>
    <externalid>0002312</externalid>
  </query>
</iq>

<!-- Response -->
<iq from='customsoperator@customsservice.domain/customsoperator' to='
    dispatcher@freightercompay.domain/QXmpp' id='qxmpp27' type='result'>
  <query xmlns='operator:iq:register'>
    <id>4</id>
  </query>
</iq>
```

The client can anytime unregister an agreement with the server. If the agreed time and all optionally sent delays has expired, the server will automatically delete the agreement.

Listing C.13: Example of a unregistration agreement request.

```
<!-- Request -->
<iq id="qxmpp27" to="customsoperator@customsservice.domain/
    customsoperator" type="get">
  <query xmlns="operator:iq:unregister">
    <id>3</id>
  </query>
</iq>

<!-- Response -->
<iq from='customsoperator@customsservice.domain/customsoperator' to='
    dispatcher@freightercompay.domain/QXmpp' id='qxmpp27' type='result'>
  <query xmlns='operator:iq:register'/>
</iq>
```

Updates to the agreement can be sent by both parties and contains the agreement id and a new updated time. An optional text message can be added to give extra information regarding the update.

Listing C.14: Example of an update to an agreement sent from a customs office to an expected truck telling him to expect an hour delay time.

```
<message from='customsoperator@customsservice.domain/customsoperator' to=
    'truck1@freightercompay.domain/QXmpp' id='rnv7I-11'>
 <updates xmlns='operator:message:updates'>
  <arrivaltime>2010-05-17T08:45:56+0200</arrivaltime>
  <text>Long queues, expect delays.</text>
  <id>4</id>
 </updates>
</message>
```

C.3. Fleet management

A very simple fleet management system where a fleet operator can assign orders to trucks and get updates about the trucks locations in almost real time. The basic element is an order which has a state of either new, assigned, accepted, rejected or completed. A pre-planned route can be attached to the order if the operator wants the truck to take a specific route. If any agreements have been set up for this job they will also be attached.

Listing C.15: Example an order being sent from a fleet operator to a truck.

```
<message from='dispatcher@freightcompany.domain/QXmpp' to='
    truck1@freightcompay.domain/QXmpp'>
  <order type='request' xmlns='nsfrits:message'>
    <route>LINESTRING(11.085205 59.500880, 11.447754 58.608334)</route>
    <id>1</id>
```

```
5       <state>assigned</state>
6       <agreements>
7         <agreement>
8           <time>2010-05-17T12:13:39+0200</time>
9           <id>4</id>
10          <servicejid>customsoperator@customsservice.domain/customsoperator
                </servicejid>
11          <clientjid>truck1@freightcompany.domain/QXmpp</clientjid>
12          <externalid/>
13          <infoid>27</infoid>
14        </agreement>
15      </agreements>
16    </order>
17  </message>
```

When an order is received by the truck the driver can choose to either accept or reject the order and sending the response back to the fleet operator.

Listing C.16: Example of a message where the truck accepts the given order.

```
1  <message from='truck1@freightcompany.domain/QXmpp' to='
       dispatcher@freightcompany.domain/QXmpp'>
2    <order type='response' xmlns='nsfrits:message'>
3      <id>1</id>
4      <state>accepted</state>
5    </order>
6  </message>
```

The truck will send its current position back to the fleet operator by certain intervals, either a time limit or a distance interval, or both. More often means that the operator gets a more updated view of the trucks, but will also increase the traffic sent of the network.

Listing C.17: Example of a truck sending its current location to the fleet operator.

```
1  <message to="dispatcher@freightcompany.domain/QXmpp">
2    <position xmlns="nsfrits:message">POINT(11.832275 58.048818)</position>
3  </message>
```

D. XML-schemas

D.1. NSFRITS Data model

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema targetNamespace="nsfrits:data" elementFormDefault="qualified"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="nsfrits:data">
3
4
5
6      <xsd:element name="infoobject">
7        <xsd:complexType>
8          <xsd:sequence>
9            <xsd:element name="validity" maxOccurs="1"
10             minOccurs="1">
11             <xsd:complexType>
12               <xsd:all>
13                 <xsd:element name="from" type="xsd:dateTime"
14                   maxOccurs="1" minOccurs="1">
15                 </xsd:element>
16                 <xsd:element name="until"
17                   type="xsd:dateTime" maxOccurs="1" minOccurs="1">
18                 </xsd:element>
19               </xsd:all>
20             </xsd:complexType>
21           </xsd:element>
22           <xsd:element name="parameterdata" maxOccurs="unbounded"
23             minOccurs="0">
24             <xsd:complexType>
25               <xsd:attribute name="type" use="required">
26                 <xsd:simpleType>
27                   <xsd:restriction base="xsd:string">
28                     <xsd:enumeration
29                       value="temperature">
30                     </xsd:enumeration>
31                     <xsd:enumeration value="length"></xsd:enumeration>
32                     <xsd:enumeration value="speed"></xsd:enumeration>
33                     <xsd:enumeration value="weight"></xsd:enumeration>
34                   </xsd:restriction>
35                 </xsd:simpleType>
36               </xsd:attribute>
37               <xsd:attribute name="value" type="xsd:float"
38                 use="required">
39               </xsd:attribute>
40             </xsd:complexType>
```

```
41            </xsd:element>
42            <xsd:element maxOccurs="1" minOccurs="0"
43              ref="datanode">
44            </xsd:element>
45            <xsd:element name="location" maxOccurs="unbounded"
46              minOccurs="0">
47              <xsd:complexType>
48                <xsd:choice>
49                  <xsd:element name="area">
50                    <xsd:simpleType>
51                      <xsd:restriction
52                        base="xsd:string">
53                        <xsd:pattern
54                          value="POLYGON\(.+\)">
55                        </xsd:pattern>
56                      </xsd:restriction>
57                    </xsd:simpleType>
58                  </xsd:element>
59                  <xsd:element name="road"
60                    type="xsd:string">
61                  </xsd:element>
62                  <xsd:element name="point">
63                    <xsd:simpleType>
64                      <xsd:restriction
65                        base="xsd:string">
66                        <xsd:pattern
67                          value="POINT\(.+\)">
68                        </xsd:pattern>
69                      </xsd:restriction>
70                    </xsd:simpleType>
71                  </xsd:element>
72                  <xsd:element name="place"
73                    type="xsd:string">
74                  </xsd:element>
75                </xsd:choice>
76              </xsd:complexType>
77            </xsd:element>
78            <xsd:element name="agreementservice" type="xsd:string"
                maxOccurs="1" minOccurs="0"></xsd:element>
79          </xsd:sequence>
80        </xsd:complexType></xsd:element>
81
82      <xsd:element name="description">
83        <xsd:complexType>
84          <xsd:sequence>
85            <xsd:element name="language" type="xsd:string" maxOccurs="1"
                minOccurs="1"></xsd:element>
86            <xsd:element name="title" type="xsd:string" maxOccurs="1"
                minOccurs="1"></xsd:element>
87            <xsd:element name="text" type="xsd:string" maxOccurs="1"
                minOccurs="0"></xsd:element>
88            <xsd:element name="sound" type="xsd:string" maxOccurs="1"
                minOccurs="0"></xsd:element>
89          </xsd:sequence>
```

```
90        </xsd:complexType></xsd:element>
91
92      <xsd:element name="datanode">
93        <xsd:complexType>
94          <xsd:sequence>
95            <xsd:element minOccurs="1" maxOccurs="1"
96              ref="description">
97            </xsd:element>
98            <xsd:element ref="datanode" maxOccurs="unbounded"
99              minOccurs="0">
100           </xsd:element>
101           <xsd:element name="attachment" maxOccurs="1" minOccurs="0">
102             <xsd:complexType>
103               <xsd:sequence>
104                           <xsd:element name="mime-type" type="
                                xsd:string">
105               </xsd:element>
106                           <xsd:element name="size" type="xsd:int">
107               </xsd:element>
108               <xsd:element name="url"
109                 type="xsd:string">
110               </xsd:element>
111             </xsd:sequence>
112           </xsd:complexType></xsd:element>
113       </xsd:sequence>
114     </xsd:complexType>
115   </xsd:element>
116
117   <xsd:element name="category">
118     <xsd:complexType>
119       <xsd:sequence>
120         <xsd:element name="id" type="xsd:string"></xsd:element>
121         <xsd:element ref="description"></xsd:element>
122       </xsd:sequence>
123     </xsd:complexType>
124   </xsd:element>
125
126   <xsd:element name="language">
127     <xsd:complexType>
128       <xsd:sequence>
129         <xsd:element name="id" type="xsd:string"></xsd:element>
130         <xsd:element name="name" type="xsd:string"></xsd:element>
131       </xsd:sequence>
132     </xsd:complexType>
133   </xsd:element>
134 </xsd:schema>
```

D.2. NSFRITS IQ

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema targetNamespace="nsfrits:iq" elementFormDefault="qualified"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="nsfrits:iq"
       xmlns:Q1="nsfrits:data">
```

```
3
4       <xsd:import schemaLocation="nsfrits-data.xsd" namespace="nsfrits:data
            "></xsd:import>
5       <xsd:element name="query">
6         <xsd:complexType>
7           <xsd:choice maxOccurs="1" minOccurs="0">
8             <xsd:all maxOccurs="1" minOccurs="0">
9               <xsd:element name="methodcall" maxOccurs="1"
10                minOccurs="1">
11                <xsd:simpleType>
12                  <xsd:restriction base="xsd:string">
13                    <xsd:enumeration
14                      value="getInfoForPoint">
15                    </xsd:enumeration>
16                    <xsd:enumeration
17                      value="getInfoForRoute">
18                    </xsd:enumeration>
19                    <xsd:enumeration value="getRouteTo"></xsd:enumeration>
20                    <xsd:enumeration value="searchPOI"></xsd:enumeration>
21                    <xsd:enumeration
22                      value="getCategories">
23                    </xsd:enumeration>
24                    <xsd:enumeration
25                      value="setCategories">
26                    </xsd:enumeration>
27                    <xsd:enumeration value="setLanguages"></xsd:enumeration
                          >
28                    <xsd:enumeration value="setRoute"></xsd:enumeration>
29                    <xsd:enumeration value="clearHistory"></xsd:enumeration
                          >
30                  </xsd:restriction>
31                </xsd:simpleType>
32              </xsd:element>
33              <xsd:element name="arguments" maxOccurs="1"
34                minOccurs="0">
35                <xsd:complexType>
36                  <xsd:sequence>
37                    <xsd:element name="argument"
38                      maxOccurs="unbounded" minOccurs="1">
39                      <xsd:complexType>
40                        <xsd:sequence>
41                          <xsd:element name="key"
42                            type="xsd:string">
43                          </xsd:element>
44                          <xsd:element name="value"
45                            type="xsd:string">
46                          </xsd:element>
47                        </xsd:sequence>
48                      </xsd:complexType>
49                    </xsd:element>
50                  </xsd:sequence>
51                </xsd:complexType>
52              </xsd:element>
53            </xsd:all>
```

```
54            <xsd:sequence maxOccurs="1" minOccurs="0">
55                      <xsd:element name="methodresponse" maxOccurs="1"
56                        minOccurs="1">
57                        <xsd:complexType>
58                          <xsd:choice>
59                            <xsd:sequence>
60                              <xsd:element ref="Q1:language"
61                                maxOccurs="unbounded" minOccurs="0">
62                              </xsd:element>
63                            </xsd:sequence>
64                            <xsd:sequence><xsd:element ref="Q1:category"
                                maxOccurs="unbounded" minOccurs="0">
65                            </xsd:element></xsd:sequence>
66                            <xsd:sequence><xsd:element ref="Q1:infoobject"
                                maxOccurs="unbounded" minOccurs="0">
67                            </xsd:element></xsd:sequence>
68                          </xsd:choice>
69                        </xsd:complexType>
70                      </xsd:element>
71            </xsd:sequence>
72          </xsd:choice>
73        </xsd:complexType>
74      </xsd:element>
75
76  </xsd:schema>
```

D.3. NSFRITS Message

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema targetNamespace="nsfrits:message" elementFormDefault="
       qualified" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="
       nsfrits:message" xmlns:p="nsfrits:data">
3
4      <xsd:import schemaLocation="nsfrits-data.xsd" namespace="nsfrits:data
           "></xsd:import>
5      <xsd:element name="position">
6        <xsd:simpleType>
7          <xsd:restriction base="xsd:string">
8            <xsd:pattern value="POINT\(.+\)"></xsd:pattern>
9          </xsd:restriction>
10       </xsd:simpleType>
11     </xsd:element>
12
13     <xsd:element name="order">
14       <xsd:complexType>
15         <xsd:annotation>
16           </xsd:annotation>
17         <xsd:sequence>
18           <xsd:element name="route" maxOccurs="1" minOccurs="0">
19             <xsd:simpleType>
20               <xsd:restriction base="xsd:string">
21                 <xsd:pattern value="LINESTRING\(.+\)"></xsd:pattern>
22               </xsd:restriction>
```

```
23              </xsd:simpleType>
24            </xsd:element>
25            <xsd:element name="id" type="xsd:string" maxOccurs="1"
26              minOccurs="1">
27            </xsd:element>
28            <xsd:element name="state" maxOccurs="1" minOccurs="1">
29              <xsd:simpleType>
30                <xsd:restriction base="xsd:string">
31                  <xsd:enumeration value="accepted"></xsd:enumeration>
32                  <xsd:enumeration value="rejected"></xsd:enumeration>
33                  <xsd:enumeration value="completed"></xsd:enumeration>
34                  <xsd:enumeration value="assigned"></xsd:enumeration>
35                  <xsd:enumeration value="new"></xsd:enumeration>
36                </xsd:restriction>
37              </xsd:simpleType>
38            </xsd:element>
39            <xsd:element ref="p:description" maxOccurs="1"
40              minOccurs="0">
41            </xsd:element>
42            <xsd:element name="agreements" maxOccurs="1"
43              minOccurs="0">
44              <xsd:complexType>
45                <xsd:sequence>
46                  <xsd:element ref="agreement"></xsd:element>
47                </xsd:sequence>
48              </xsd:complexType>
49            </xsd:element>
50          </xsd:sequence>
51          <xsd:attribute name="type" use="required">
52            <xsd:simpleType>
53              <xsd:restriction base="xsd:string">
54                <xsd:enumeration value="request"></xsd:enumeration>
55                <xsd:enumeration value="response"></xsd:enumeration>
56              </xsd:restriction>
57            </xsd:simpleType>
58          </xsd:attribute>
59        </xsd:complexType>
60      </xsd:element>
61
62      <xsd:element name="updates">
63        <xsd:complexType>
64          <xsd:sequence>
65            <xsd:element ref="p:infoobject"></xsd:element>
66          </xsd:sequence>
67        </xsd:complexType>
68      </xsd:element>
69
70      <xsd:element name="alert" type="xsd:string"></xsd:element>
71
72      <xsd:element name="agreement">
73        <xsd:complexType>
74          <xsd:sequence>
75            <xsd:element name="id" type="xsd:string" maxOccurs="1"
76              minOccurs="1">
```

```
77            </xsd:element>
78            <xsd:element name="time" type="xsd:dateTime"
79              maxOccurs="1" minOccurs="1">
80            </xsd:element>
81            <xsd:element name="jid" type="xsd:string" maxOccurs="1"
82              minOccurs="1">
83            </xsd:element>
84            <xsd:element name="externalid" type="xsd:string"
85              maxOccurs="1" minOccurs="0">
86            </xsd:element>
87            <xsd:element name="infoid" type="xsd:string" maxOccurs="1"
                  minOccurs="1"></xsd:element>
88          </xsd:sequence>
89        </xsd:complexType>
90      </xsd:element>
91  </xsd:schema>
```

D.4. NSFRITS Agreement

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <xsd:schema targetNamespace="nsfrits:agreement" elementFormDefault="
        qualified" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="
        nsfrits:agreement">
3
4       <xsd:element name="query">
5         <xsd:complexType>
6           <xsd:choice>
7             <xsd:sequence>
8               <xsd:element name="agreetime" type="xsd:dateTime"
9                 maxOccurs="1" minOccurs="1">
10              </xsd:element>
11              <xsd:element name="externalid" type="xsd:string" maxOccurs="1
                    " minOccurs="0"></xsd:element>
12              <xsd:element name="jid" type="xsd:string" maxOccurs="1"
                    minOccurs="0"></xsd:element>
13            </xsd:sequence>
14            <xsd:sequence>
15              <xsd:element name="id" type="xsd:string" maxOccurs="1"
                    minOccurs="1"></xsd:element>
16            </xsd:sequence>
17          </xsd:choice>
18        </xsd:complexType>
19      </xsd:element>
20
21      <xsd:element name="updates">
22        <xsd:complexType>
23          <xsd:sequence>
24            <xsd:element name="arrivaltime" type="xsd:dateTime" maxOccurs="
                  1" minOccurs="1"></xsd:element>
25            <xsd:element name="text" type="xsd:string" maxOccurs="1"
                  minOccurs="0"></xsd:element>
26            <xsd:element name="id" type="xsd:string" maxOccurs="1"
                  minOccurs="1"></xsd:element>
```

```
27          </xsd:sequence>
28        </xsd:complexType>
29      </xsd:element>
30  </xsd:schema>
```

D.5. Fleet management

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema targetNamespace="nsfrits:fleet" elementFormDefault="qualified
       " xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="nsfrits:fleet">
3
4      <xsd:element name="order">
5        <xsd:complexType>
6          <xsd:choice>
7            <xsd:element name="route" maxOccurs="1" minOccurs="0">
8                      <xsd:annotation>
9                        <xsd:documentation>Optional preplanned route.</
                             xsd:documentation>
10                     </xsd:annotation>
11                     <xsd:simpleType>
12                 <xsd:restriction base="xsd:string">
13                   <xsd:pattern
14                     value="LINESTRING\((\d+\.\d*\s\d+\.\d*(,)?\s*)+\)">
15                   </xsd:pattern>
16                 </xsd:restriction>
17             </xsd:simpleType>
18           </xsd:element>
19           <xsd:element name="id" type="xsd:string" maxOccurs="1"
                 minOccurs="1">
20             <xsd:annotation>
21               <xsd:documentation>Order id.</xsd:documentation>
22             </xsd:annotation></xsd:element>
23           <xsd:element name="state" maxOccurs="1" minOccurs="1">
24                     <xsd:annotation>
25                       <xsd:documentation>The state of the order, can be
                             either new, assigned, accepted, rejected or
                             completed.</xsd:documentation>
26                     </xsd:annotation>
27                     <xsd:simpleType>
28                 <xsd:restriction base="xsd:string">
29                   <xsd:enumeration value="new"></xsd:enumeration>
30                   <xsd:enumeration value="assigned"></xsd:enumeration>
31                   <xsd:enumeration value="accepted"></xsd:enumeration>
32                   <xsd:enumeration value="rejected"></xsd:enumeration>
33                   <xsd:enumeration value="completed"></xsd:enumeration>
34                 </xsd:restriction>
35             </xsd:simpleType>
36           </xsd:element>
37           <xsd:element name="agreements" maxOccurs="1"
38             minOccurs="0">
39                     <xsd:annotation>
40                       <xsd:documentation>Set of agreements associated
                             with this order.</xsd:documentation>
```

```
41                    </xsd:annotation>
42                    <xsd:complexType>
43                <xsd:sequence>
44                  <xsd:element name="time" type="xsd:dateTime"
45                    maxOccurs="1" minOccurs="1">
46                                  <xsd:annotation>
47                                    <xsd:documentation>Agreement time</
                                          xsd:documentation>
48                                  </xsd:annotation>
49                  </xsd:element>
50                  <xsd:element name="id" type="xsd:string" maxOccurs="1"
                        minOccurs="1">
51                                  <xsd:annotation>
52                                    <xsd:documentation>Agreement id</
                                          xsd:documentation>
53                                  </xsd:annotation>
54                  </xsd:element>
55                  <xsd:element name="servicejid"
56                    type="xsd:string" maxOccurs="1" minOccurs="1">
57                                  <xsd:annotation>
58                                    <xsd:documentation>Jid for the service
                                          in the agreement</xsd:documentation>
59                                  </xsd:annotation>
60                  </xsd:element>
61                  <xsd:element name="clientjid"
62                    type="xsd:string" maxOccurs="1" minOccurs="1">
63                                  <xsd:annotation>
64                                    <xsd:documentation>Jid for the client
                                          in the agreement</xsd:documentation>
65                                  </xsd:annotation>
66                  </xsd:element>
67                  <xsd:element name="externalid"
68                    type="xsd:string" maxOccurs="1" minOccurs="0">
69                                  <xsd:annotation>
70                                    <xsd:documentation>External id for use
                                          by external services</
                                          xsd:documentation>
71                                  </xsd:annotation>
72                  </xsd:element>
73                  <xsd:element name="infoid"
74                    type="xsd:string" maxOccurs="1" minOccurs="1">
75                                  <xsd:annotation>
76                                    <xsd:documentation>Id for the
                                          associated InfoObject</
                                          xsd:documentation>
77                                  </xsd:annotation>
78                  </xsd:element>
79                </xsd:sequence>
80              </xsd:complexType>
81            </xsd:element>
82
83          </xsd:choice>
84          <xsd:attribute name="type">
85                  <xsd:annotation>
```

```
86                    <xsd:documentation>Type of order message, either
                          request to the client or response back to the
                          operator.</xsd:documentation>
87                  </xsd:annotation>
88                  <xsd:simpleType>
89              <xsd:restriction base="xsd:string">
90                <xsd:enumeration value="request"></xsd:enumeration>
91                <xsd:enumeration value="response"></xsd:enumeration>
92              </xsd:restriction>
93            </xsd:simpleType>
94          </xsd:attribute>
95        </xsd:complexType>
96      </xsd:element>
97  </xsd:schema>
```