CHALMERS

Evaluating GST using wireless off-board diagnostics

- a Global System for Telematics project

Master of Science Thesis in computer science

MIKAEL JOHANSSON
PATRICK STENBERG
ROBERT LAXING

Computer Science and Engineering Program
CHALMERS UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Division of Computer Engineering
Göteborg 2006

Abstract

This is a master thesis report done at Chalmers university of technology in cooperation with Volvo Technology Corporation (VTEC). The purpose of this thesis was to evaluate whether the GST architecture can solve the problems with long production cycles and lack of compatibility between services. The GST project is an Integrated Project within EU's 6th Framework Programme, whose vision is to create an open environment in which innovative telematics services can be developed and delivered. The GST architecture is based on OSGi. The evaluation was done by implementing a service using the GST architecture. Prior to doing this evaluation, an analysis of different services was conducted, from which we decided to implement a diagnostics service.

Our solution enables retrieval of diagnostic trouble codes, logging of vehicle data and presentation of collected information in an application at the back-office. We have showed that it is possible to develop a service in a very short time using the GST architecture. The possibility of increased sharing of code between different types of vehicles (cars, trucks, boats) can lead to increased efficiency for VTEC and AB Volvo. However, the architecture is quite complex which puts large requirements both on hardware and developers. VTEC has granted further work on the implemented service, outside of the scope of this work.

Keywords: GST, Global System for Telematics, Pocket PC, Java, VTEC, Volvo Technology Corporation, diagnostics, uptime management, RVD, OSGi

Sammanfattning

Denna rapport är en del av ett examensarbete utfört på Chalmers tekniska högskola i samarbete med Volvo Technology Corporation (VTEC). Syftet var att utvärdera om GST-arkitekturen kan lösa problemen med långa produktionscykler och kompabilitet mellan tjänster. GST är ett projekt inom EU:s sjätte ramprogram vars mål är att skapa en öppen miljö för utveckling och leverans av telematiktjänster. För att utvärdera GST, som är baserad på OSGi, implementerades en tjänst. Valet av tjänst baserades på en analys där olika tjänsters egenskaper behandlades.

Vår tjänst gör det möjligt att läsa diagnostikkoder, logga fordonsdata och presentera insamlad information i en kontorsapplikation. Vi har visat att det är möjligt att utveckla en tjänst med hjälp av GST-arkitekturen under en kort tidsperiod. VTEC och AB Volvo bör kunna öka effektiviteten genom att utnyttja möjligheten att dela kod mellan olika typer av fordon (bilar, lastbilar, båtar). Däremot är arkitekturen relativt komplex vilket innebär höga krav på både hårdvara och utvecklare. VTEC har beviljat ytterligare arbete på den implementerade tjänsten. Detta arbete kommer utföras efter examensarbetets avslutande.

Nyckelord: GST, Global System for Telematics, Pocket PC, Java, VTEC, Volvo Technology Corporation, diagnostik, uptime management, RVD, OSGi

Preface

This is a thesis work done at the department of Computer Engineering at Chalmers University of Technology supervised by associate professor Tomas Olovsson, in cooperation with Volvo Technology Corporation supervised by project manager Mats Örblom.

We have used the OSGi frameworks UbiServ and Knopflerfish provided by Gatespace. The hardware platform used in the vehicle is an HP iPAQ h5450 running Microsoft Pocket PC 4.20.1081/13100. The JVM used is IBM J9. Another HP iPAQ, h6340 is used as a GPRS modem.

Our back-end server is based on Apache 2, Tomcat 5 and Axis 1, using MySQL 14.7. Redhat Enterprise Linux 3.0AS running on a HP ProLiant DL380.

The development tools used are Eclipse v 3.1, Microsoft Embedded Visual C++ 4.0 and JDK 1.4.2 for J2ME and J2EE.

Our solution uses J1587 interface software and hardware from Volvo Parts / IT University of Göteborg. The software J1587 Navigator from Volvo 3P department 26300 was used for development and testing.

Report structure

The first chapter contains definitions and abbreviations used in this report. Chapter 2 provides the reader with information about the background and the purpose of this thesis work. It also describes the delimitations, i.e. subjects that we did not include in the work. Different methods and models that we used are then presented. How did we plan the thesis work? How did we implement the service? How did we divide the work between the three of us? Questions like these will be answered in the method chapter. In the following chapter, we analyse our thesis work. What kind of service did we choose? What software and hardware were used and how were the different parts implemented? The Conclusions and recommendations chapter describes the conclusions of this thesis work and provide recommendations and possible future work for the interested audience. Chapters 7 and 8 contain information about the references used and recommended literature.

Acknowledgments

Mats Örblom, our supervisor at VTEC. Marcus Larsson, VTEC system developer. Tomas Olovsson at Chalmers University of Technology. Carl Johan Andersson at Volvo Parts. Jonas Kuschel at the IT University of Göteborg. Lars-Gunnar Olsson, Janne Andersson and Denny Rönnberg at Volvo truck center Bäckebol. Catharina Voldstedtlund at Open Arena Lindholmen. We were allowed to do all the work on-site at VTEC, which gave us access to a nice environment and good resources for information (library, co-workers).

Table of contents

1. Glossary 1
   1.1. Definitions 1
   1.2. Abbreviations 1
2. Introduction 2
   2.1. Background 2
   2.2. Purpose 2
   2.3. Delimitations 3
   2.4. Existing solutions 3
      2.4.1. BMW iDrive 3
      2.4.2. Scania Fleet Management 3
      2.4.3. Dynafleet / Dynafleet Online 4
      2.4.4. OnStar® 4
   2.5. GST 4
      2.5.1. GST basics 4
      2.5.2. The management layer 6
         2.5.2.1.Steering Committee (SC) 6
         2.5.2.2.Core Team (CT) 6
         2.5.2.3.Core Architecture Group (CAG) 7
      2.5.3. The subprojects 7
         2.5.3.1.Open systems (OS) 7
         2.5.3.2.Certifications (CERTECS) 7
         2.5.3.3.Service Payment (S-PAY) 7
         2.5.3.4.Safety Channel (SAF-CHAN) 8
         2.5.3.5.Rescue (RSQ) 8
         2.5.3.6.Enhanced Floating Car Data (EFCD) 8
         2.5.3.7.Security (SEC) 9
      2.5.4. Subprojects useful for our solution 9
      2.5.5. OSGi 10
3. Method 13
4. Analysis 15
   4.1. SWOT analysis of GST 15
      4.1.1. GST for Volvo truck brands 16
      4.1.2. GST for VTEC 16
      4.1.3. GST for the customers 16
   4.2. Service analysis 17
      4.2.1. Motivation for diagnostics 17
      4.2.2. Diagnosis workflow at Volvo Truck Centre Bäckebol 18
   4.3. Service scenario 18
   4.4. System overview 19
   4.5. Implementation 20
      4.5.1. Connectivity 20
         4.5.1.1.TCU ⇔ J1587 diagnostics bus 20
         4.5.1.2.TCU ⇔ Internet 21
         4.5.1.3.Back-end 22
         4.5.1.4.Back-office 22
      4.5.2. Communication protocols 22
         4.5.2.1.Back-office → back-end 22
         4.5.2.2.Back-end → TCU 23
         4.5.2.3.TCU → back-end 23
      4.5.3. Hardware 23

4.5.3.1.Back-end....23
4.5.3.2.TCU....23
4.5.4. Software....24
4.5.4.1.In-vehicle software....24
4.5.4.2.Back-office system....28
4.5.4.3.Back-end....33
4.6. Service Submission Contest....36
5. Conclusions and recommendations....37
6. Future work....38
7. References....39
8. Literature....41
9. Appendix A – Service requirements specification....I
10. Appendix B – Service analysis table....II
11. Appendix C – Detaljplan förlängning....III

List of tables

Table 1 – A bundle's different states....11
Table 2 – SWOT analysis....15
Table 3 – Update frequency of panels....30
Table 4 – Installed software at the back-end....34

List of figures

Figure 1 – Overview of GST structure [GSTWWW]....6
Figure 2 – Open Systems service overview [GSTPRE]....9
Figure 3 – A bundle's different states [WEBSPH]....10
Figure 4 – The OSGi architecture [WEBSPH]....12
Figure 5 – Project time plan....13
Figure 6 – The J1587 interface jacket and the TCU....21
Figure 7 – IP-based network connectivity overview....22
Figure 8 – Steps for creating an application that uses JNI [JNISTEPS]....25
Figure 9 – Gatespace's Knopflerfish....26
Figure 10 – The framework running on Pocket PC....27
Figure 11 – Our test application....28
Figure 12 – GUI structure....30
Figure 13 – Info tab....31
Figure 14 – Notes tab....31
Figure 15 – Error codes tab....32
Figure 16 – Request log tab....32
Figure 17 – View log tab....33

1. Glossary

These sections provide definitions for terms and abbreviations used in this document.

1.1. Definitions

Off-board system	Generic term referring to back-end system.
On-board system	Generic term referring to the in-vehicle system (hardware and software).
Back-end	The infrastructure of servers, databases, and software that supports the actions of users who interact with delivered content.
Back-office	User interface, usually run at a company's office, which enables interaction with the back-end.
TCU	In-vehicle unit with communications capabilities on which the in-vehicle software is run.

1.2. Abbreviations

CAN	Controller Area Network
DLL	Dynamic Link Library
DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit
GPRS	General Packet Radio Service
GST	Global System for Telematics
GUI	Graphical User Interface
HLA	High Level Architecture
HMI	Human Machine Interface
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol (network communications) OR Integrated project (GST)
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JNI	Java Native Interface
JVM	Java Virtual Machine
LSP	Lindholmen Science Park
MID	Message Identifier
MPPE	Microsoft Point to Point Encryption
OAL	Open Arena Lindholmen
PID	Parameter Identifier
PPID	Proprietary Parameter Identification
PPTP	Point-to-Point Tunneling Protocol
RPC	Remote Procedure Call
RVD	Remote Vehicle Diagnostics
SAE	Society of Automotive Engineers
SMS	Short Message Service
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SWOT	Strengths, Weaknesses, Opportunities, and Threats
TCP	Transmission Control Protocol
TCU	Telematics Control Unit
UDP	User Datagram Protocol
V3P	Volvo Purchasing, Product Development and Product Planning
VAS	Volvo Action Services
VCC	Volvo Car Corporation
VPN	Virtual Private Network
VTC	Volvo Truck Corporation
VTEC	Volvo Technology Corporation

2. Introduction

2.1. Background

Volvo and Renault Trucks are today offering a wide variety of services to the trucking industry. These services are e.g. financing, insurance, Dynafleet, view vehicle specification and driver development programs. Some of the services are offered online through a web portal and require communication with the vehicle. Rapid development within the telecom industry and the growth of third party services are putting new requirements on the vehicle manufacturers. The vehicle manufacturers have product cycles from around five to seven years while the telecom industry has a product cycle of about one year. In order to be able to develop and sell new services the vehicle industry examines the possibility of downloading and updating both old and completely new service concepts to the vehicle on a wireless basis.

Today the development of a new telematics service is based on a full system development, including both software and hardware. This is how it works for e.g. Dynafleet and it is a very time consuming and cost-ineffective process. Every time a new function in the service is added, the whole system must be updated and verified. To make the development of new telematics services more cost-effective and to increase the availability of services, there must be a standardized way to develop and deploy new services. This is why Global System for Telematics (GST) was formed.

Volvo is among other partners involved in GST, which is an EU-funded Integrated Project [GSTWWW] that is creating an open and standardized end-to-end architecture for automotive telematics services. The purpose of GST is to create an environment based on open standards to increase the market for on-line services and to benefit the development of safety services for the car industry. The vision of GST is that all future vehicles will be equipped with various communications means to interact with each other and provide an environment based on a common architecture and standard interfaces.

GST is based on Open Services Gateway initiative (OSGi), which is a framework for installing and managing standardized software components. The framework also provides an environment where these components can interact according to common rules.

2.2. Purpose

The purpose of this work was to evaluate whether the GST platform can solve the problems with long production cycles and lack of compatibility between services. In order to do this, a service was chosen and implemented according to the GST specifications. Service selection was based on what type of service Volvo Truck Corporation (VTC) wanted to provide, what we found useful for the truck driver/owner and the technical prerequisites. We investigated how much functionality could be added during a short (a few weeks) implementation phase. We also investigated how to transfer data and what communication method to use. An analysis was made in order to study the usefulness of GST for VTC, Volvo Technology Corporation (VTEC) and vehicle users.

The objectives for this thesis are to:

- Design a platform-independent service
- Develop a competitive GST-compliant service
- Verify that the length of the production cycle for such a project can be sufficiently shorter than today.
- Research the possibilities to take part in the GST service submission contest [GSTWWW] that lets us test the service against other EU developers.

2.3. Delimitations

- No vehicle to vehicle-communication. All communication between back-office and the vehicle is done through the back-end.
- Only a prototype will be made.
    - Vehicle data and interfaces will be emulated if necessary.
    - Functionality tests will be made, but no full product verification will be done.
- The service will be directed towards the commercial truck market. No attempt will be made to make the service work on cars.
- No analysis of the logging data will be done. It is up to the user to interpret the result and take appropriate action.
- The service will not provide any means of interaction with the driver, nor will it be directed toward the truck owner.
- Authentication and authorization will not be implemented.
- We will not attempt to make the software work on telematics units installed in today's vehicles; testing and development will be done for iPAQ only.

2.4. Existing solutions

Many companies are interested in providing telematics services to vehicle owners. Below follows a brief description of a few existing telematics solutions. All of these are vendor-specific, i.e. they work only in trucks manufactured by the vendor.

2.4.1. BMW iDrive

BMW uses an OSGi implementation from ProSyst for development and deployment of applications to the vehicle infotainment platform called iDrive [PROSYST]. The difference from GST is that the OSGi specification does not specify how to communicate between nodes and how provisioning and deployment is done. The solution is a closed system and it is not possible for customers to use services from other service providers.

2.4.2. Scania Fleet Management

Scania Fleet Management uses two different platforms; one based on Pocket PC and one based on regular PC's running Microsoft Windows XP Embedded [SCANIA]. On the latter, it is possible for customers to install their own Windows applications.

2.4.3. Dynafleet / Dynafleet Online

In Europe, Volvo offers Dynafleet, which they claim is "the market's only turnkey transport information system" [VOLVOWEB]. It can be integrated with Dynafleet Online, a web portal that makes it possible to retrieve vehicle information and send/receive text messages directly in a web browser. It is not possible for customers to install new services on their own.

2.4.4. OnStar®

General Motors (GM) has developed a system called OnStar®. The system provides a lot of functionality, but is not based on an open standard. Almost all new GM vehicles are pre-equipped with the OnStar® system and over two million users are subscribed [ONSTAR]. From the beginning OnStar® was a safety system providing functionality for road and emergency assistance but the functionality has recently been extended to include other telematics services such as Remote Vehicle Diagnostics (RVD). Since OnStar® is a closed system it is not possible for the customers to choose other service providers for their vehicle.

2.5. GST

Since the purpose of this work is to evaluate GST, a basic knowledge about GST is necessary. This section will briefly explain what GST is all about, what the vision is and how the organisation works.

2.5.1. GST basics

Within the European Union's 6th Framework Programme [EUFP6] for research and technical development there is a new instrument used called Integrated Projects (IPs). The IPs are used for mobilising the critical mass of activities and resources needed to achieve ambitious scientific and technical objectives. The GST project is one of those IPs. [GSTWWW]

The vision of GST is to create an open environment in which innovative telematics services can be developed and delivered cost-efficiently. The goal is to increase the number of telematics services available to manufacturers and consumers. The GST project was started in March 2004 and will last until February 2007. It has a budget of € 21.5 million and more than 50 companies involved.

GST essentially consists of seven subprojects (SPs) that work together towards a common purpose. The subprojects can be divided into two groups: Technology-oriented and service-oriented (see *Figure 1*). The subprojects, which will be explained more in 2.5.3, are: [GSTWWW]

Technology-oriented:

- Open Systems
- Certification
- Service Payment
- Security

Service-oriented:

- Rescue
- Enhanced Floating Car Data
- Safety Channel

The SPs are intended to support a specific area of knowledge and to work towards a subset of the GST specification. To make it possible to reach a high level of integration between the SPs within GST a management layer has been created (see *Figure 1*). The management layer includes three groups: [GSTQTY]

- The Steering Committee (SC)
- The Core Team (CT)
- The Core Architecture Group (CAG)

The groups of the management layer will be described briefly in 2.5.2.

The GST project aims to achieve high level of integration, but to make all SPs dependant on the integration results achieved by the three groups in the management layer represents an unacceptable risk. To avoid the risks of showstoppers in the SPs due to the co-ordination of the groups in the management layer the so-called "Principle of independence" was formed:

"All SPs have been set-up in such a way that all SP deliverables and milestones can be achieved in an independent way. The SP managers are solely responsible for achieving SP results. It is their choice whether or not to use the results of the work of the special entities that have been set-up for project co-ordination purposes." [GSTQTY]

To break down the work into manageable structures and project phases work has been divided into seven work packages (WPs). The WPs has specific targets, which makes it easier to mobilise specific competences. The WPs are:

- WP 1 – Project management
- WP 2 – Use cases and system requirements
- WP 3 – Architecture and interface specifications
- WP 4 – Prototype development
- WP 5 – Field trials
- WP 6 – Validation
- WP 7 – Dissemination and exploitation

In WP 4 and WP 5, a prototype will be developed and tested. To be able to carry out an end-to-end implementation and to test the specifications, seven test sites (TSs) have been created [GSTWWW]. The TSs are located in: [GSTQTY]

- Aachen-Rüsselsheim
- Gothenburg
- Munich
- London
- Paris
- Torino
- Stuttgart

The TSs are involved in one or more subproject. More information about which SPs the TSs are involved in can be found in 2.5.3.

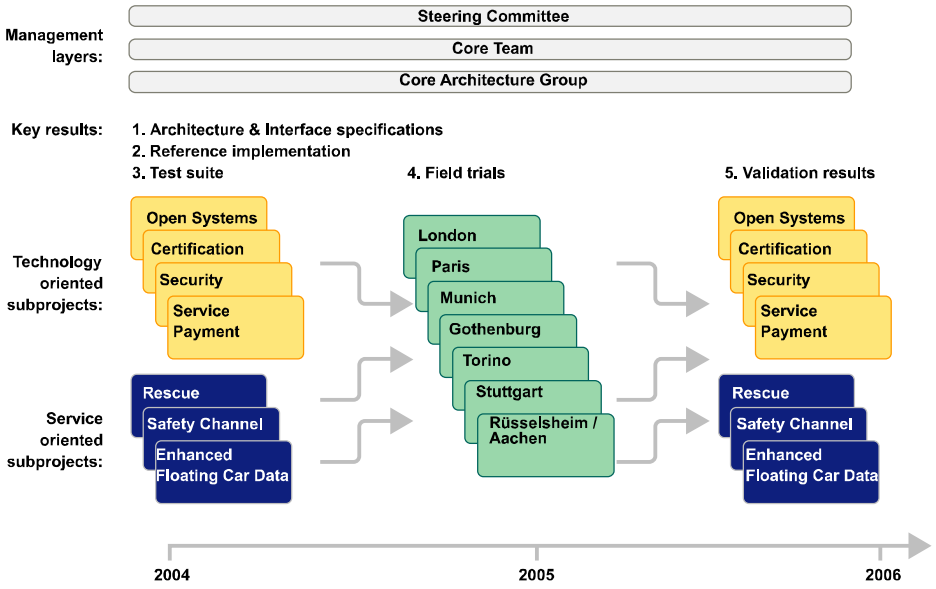


***Figure 1* – Overview of GST structure [GSTWWW]**

2.5.2. The management layer

The management layer is responsible for the horizontal activities, i.e. make it possible to keep a high integration level between the SPs. The management layer contains three groups: Steering Committee (SC), Core Team (CT) and Core Architecture Group (CAG). These groups will be described in 2.5.2.1 - 2.5.2.3. Information can be found in [GSTWWW] and [GSTQTY] .

2.5.2.1. Steering Committee (SC)

Mission statement:
"The SC has the overall responsibility for the project, for reviewing progress and for steering activities towards objectives and obligations" [GSTQTY] .

The SC's tasks are to handle the management of the entire project and monitor the progress. It should also make sure that the project proceeds in the right direction and endorse the standards and guidelines.

The members are:
Renault (Chairman), BMW (Vice-chairman), Bosch, DaimlerChrysler, Deutsche Telekom/T-Systems, ERTICO, Fiat, Ford, France Télécom/Orange, Mondial Assistance/Allianz, Motorola, Opel, Siemens VDO, SRA, Telecom Italia and Volvo.

2.5.2.2. Core Team (CT)

Mission statement:
"The CT is responsible for the overall operational management of the project. Its task is to ensure that agreed quality targets, synergies and deadlines at IP-level, sub-project and test site level are met." [GSTQTY]

The CT´s task is in other words to manage the day-to-day business of the project.

The members are:
The SP Managers and the TS Managers

2.5.2.3. Core Architecture Group (CAG)

Mission statement:
"The CAG is responsible for articulating, maintaining, extending and communicating the GST vision. Its task is to unify the philosophy, methodology, architecture and toolchain used to create the GST." [GSTQTY]

Among the tasks of the CAG can be found that the CAG should ensure consistency across the different SPs and to monitor the progress of the SPs. It should also make sure that the test site implementations follow the GST standards.

The members are:
ADSE, BMW, Bosch, DaimlerChrysler, ERTICO, France Telecom, Gatespace, Prosyst, Renault, Siemens VDO, Technical University of Munich, Telcordia (Chief Architect), TNO and T-Systems.

2.5.3. The subprojects

In this section the seven SPs, mentioned above, will be briefly explained.

2.5.3.1. Open systems (OS)

To ensure interoperability, OS aims to enable end users to use services from different locations and vehicles. This is done by standardising key interfaces and consolidating requirement from the other subprojects (see *Figure 2*). For more information about OS see section 2.5.4.

OS trials will take place at the following sites:

- Aachen-Rüsselsheim
- Gothenburg
- Munich
- Paris
- Stuttgart
- Turin

The following partners are involved:
ADSE, BMW, Bosch, DaimlerChrysler, France Telecom, Gatespace, Motorola, ProSyst, Renault, Siemens VDO, Telcordia, TNO, T-Systems and TUM

2.5.3.2. Certifications (CERTECS)

The task for the CERTECS subproject is to specify, prototype and validate a certification process for telematics components, systems and services [GST-CERTECS].

The trials will take place at the following test sites:

- Munich
- Torino
- Paris

The following partners are involved:

TÜV Rheinland Group, CETECOM, ETSI, Fiat CRF, Ford, JTEST, NAVTEQ, PTV, Renault, TRIALOG, TÜV, Telcordia.

2.5.3.3. Service Payment (S-PAY)

S-PAY is supposed to define common, transparent and reliable payment mechanisms. These mechanisms can then be utilized by service developers for easy electronic billing for telematics services.

The trials will take place at the following test sites:

- Aachen-Rüsselsheim
- Paris

The following partners are involved in the subproject:

FT R&D, Orange, ProSyst, PTV, Q-Free, Renault, TNO and Trusted Logic

2.5.3.4. Safety Channel (SAF-CHAN)

The goal for the SAF-CHAN subproject is to improve both safety and mobility on the road. This is accomplished by making it possible to alert travellers of incidents, events and current traffic status.

The trials will take place at the following test sites:

- Aachen-Rüsselsheim
- Gothenburg
- Munich

The following partners are involved in the subproject:

ERTICO, BMW, EBU, GEWI, Navteq, PTV, Robert Bosch, SES Global, Siemens VDO, SRA, TDF, Tele Atlas and TNO

2.5.3.5. Rescue (RSQ)

Enabling faster response from authorities after an incident has occurred is the vision of RSQ. Correct and up-to-date information should be delivered to authorities regardless of type of vehicle and where the incident occurred.

The trials will be performed at the following test sites:

- Aachen-Rüsselsheim
- London
- Munich
- Torino

The following partners are involved in the subproject:

ERTICO, Fiat CRF, France Telecom, Kreis Offenbach, Mizar,
Motorola, Orange, Petards, Sussex Police, TNO and Volvo.

2.5.3.6. Enhanced Floating Car Data (EFCD)

The mission of the EFCD subproject is to increase the number of in-car sensors in order to increase the available safety-relevant data and provide better event detection in case of an incident. A framework is defined, which enables efficient propagation of data.

The trials will be performed at the following test sites:

- Aachen-Rüsselsheim
- Paris
- Turin

The following partners are involved in the subproject:
PTV, Fiat CRF, Ford, ProSyst, Renault, TDF and TNO.

2.5.3.7. Security (SEC)

By providing functions and infrastructure for secure telematics applications, the SEC subproject provides a base that supports high requirements on security, privacy and reliability.

The trials will be performed at the following test sites:

- Munich
- Paris
- Stuttgart

The following partners are involved in the subproject:

Trialog, BMW, DaimlerChrysler, KUL, Renault and TUM

2.5.4. Subprojects useful for our solution

Since the functionality provided by the OS subproject could be useful in this thesis work, an extended description is provided here.

The structure of the subproject follows the High Level Architecture (HLA) defined within the GST project. In *Figure 2* the HLA is shown from the OS subproject's point of view. The concept of the architecture is that a service provider develops a service and sets up a service centre. The service centre can be a server providing information about e.g. traffic or weather. The service provider also deploys a service application to a control centre from where the application can be downloaded by a client system. The control centre provides some kind of portal from where the user of the client system can browse the available services and download the wanted service, which will be installed on the client system. When installed the service connects to the service centre and download the information needed.

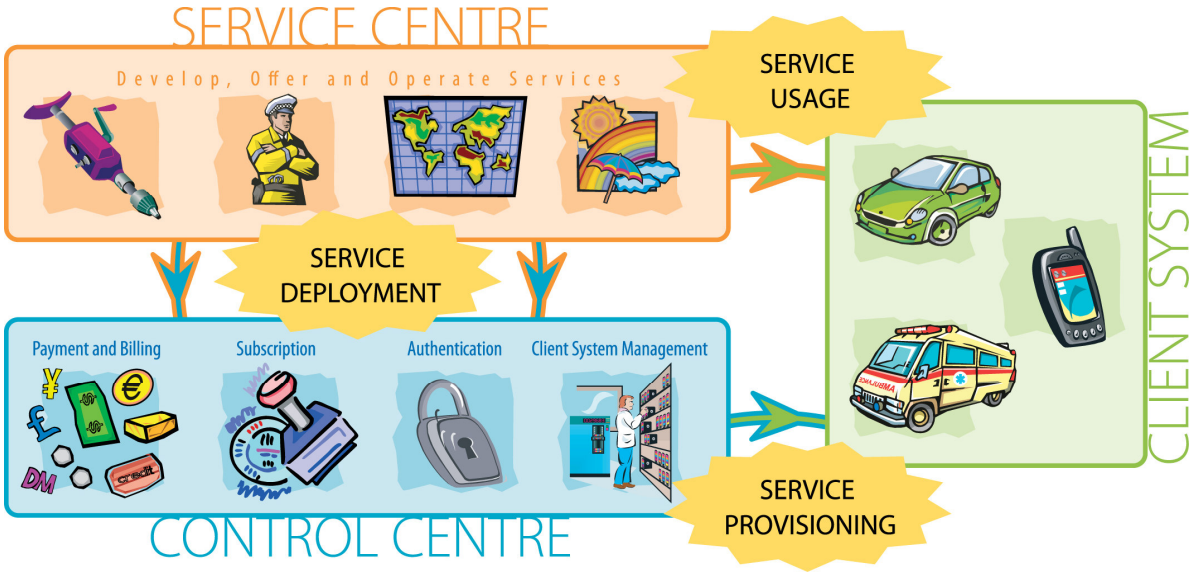


***Figure 2* – Open Systems service overview [GSTPRE]**

The control centre should keep records over the users and make sure that only subscribed users can use the services. It should also handle the payment of the services.

Another subproject that could be useful in our project is the SEC subproject. This subproject provides specifications about authentication and security for the communication. Further information about SEC can be found in [GSTWWW].

2.5.5. OSGi

The following information is a summary. A more detailed description of OSGi can be found in [OSGI] and [WEBSPH].

GST is using the OSGi Service Gateway as its execution environment. The specification of the OSGi Service Gateway defines a standardized and component oriented architecture for software components called *bundles*. Software applications consist of one or several bundles collaborating with each other. They are connected only through a *service interface*. Therefore, developers only need to deal with the service interfaces; they do not need to know anything about how the bundles are implemented. The OSGi Service Gateway, often called the *framework*, is the part of the platform that is responsible for installing, removing and updating bundles. It also handles the registration/unregistration of services, i.e. a service can only use other services that have been registered in the framework.

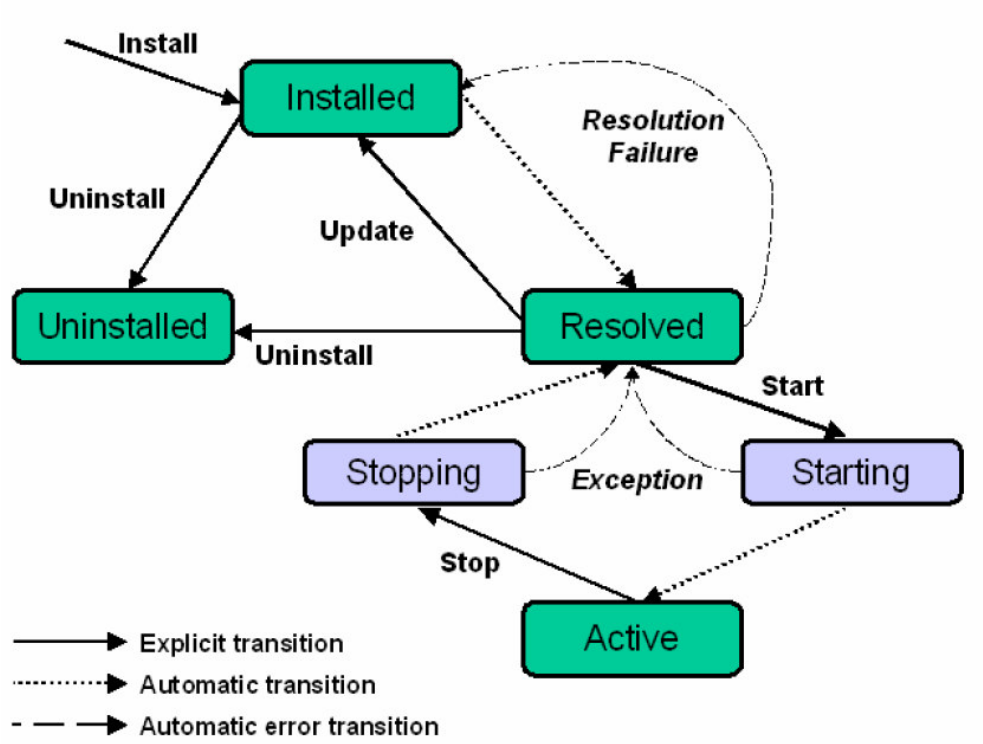


***Figure 3* – A bundle's different states [WEBSPH]**

A bundle is always in a certain state (see *Figure 3*). *Table 1* briefly describes these states and what they mean.

Installed state	The bundle is loaded into memory
Resolved state	The framework has resolved the bundle's package dependencies
Starting state	If the resolving was successful, the bundle will start by executing the bundle's start method
Active state	If the starting process was successful, the bundle will be in an active state
Stopping state	The framework is trying to stop the bundle by executing the bundle's stop method
Uninstalled state	The bundle has been uninstalled

***Table 1* – A bundle's different states**

By installing a framework to a networked device, it becomes possible to manage the life cycle of the software from anywhere in the network. Bundles can be installed or removed directly while the software is running without any disruption.

A *service* is a Java class or an interface that describes the methods and variables the implementation of the service must provide. It is possible to create a service using a Java class but [WEBSPH] recommends creating a service using a Java interface instead.

A *bundle* is simply a JAR (Java archive) file. A JAR file is a set of files that are grouped together and compressed with the ZIP (Zone information protocol) file format, making them faster to download. It always contains a so-called manifest-file that describes the bundle, what it provides and what other services it imports and exports.

A *bundle activator* is a class responsible for initializing its' bundle when the bundle starts and for cleaning up after the bundle when it stops.

A *bundle context* is a unique identifier that the bundle uses to communicate with the OSGi framework. This enables e.g. the bundle to use services that other bundles provide.

Figure 4 shows the different layers in the OSGi architecture.

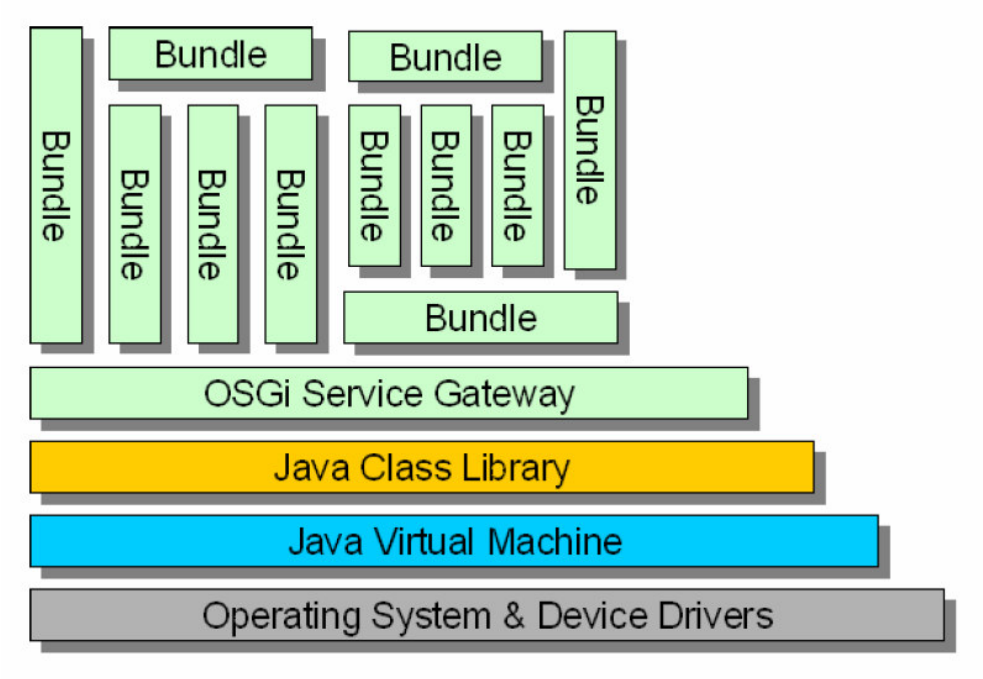


Figure 4 **– The OSGi architecture [WEBSPH]**

3. Method

The research approach was qualitative; to get inspiration in order to develop a competitive and useful service, we studied a small number of existing services, in order to either develop an existing service in a new and better way or to develop an entirely new service.

We visited Volvo Truck Center in Bäckebol, where we gathered information about how diagnostics is used in their work today. A visit to Volvo Parts gave us valuable information about how they look upon diagnostics and how diagnostics is performed on boats using their product "Vodia". A Strengths, Weaknesses, Opportunities, and Threats (SWOT) analysis of GST was performed in order to investigate pros and cons using GST for Volvo truck brands, VTEC and vehicle owners.

A time plan for the project was constructed (see *Figure 5*), dividing the work into several sub-tasks. We decided that some work on the final report should be done every week and time was reserved for performing final tests and demonstration of the solution. A software requirement specification was written, and this specification was the base for the implementation. Function testing was done every time a new function had been implemented.

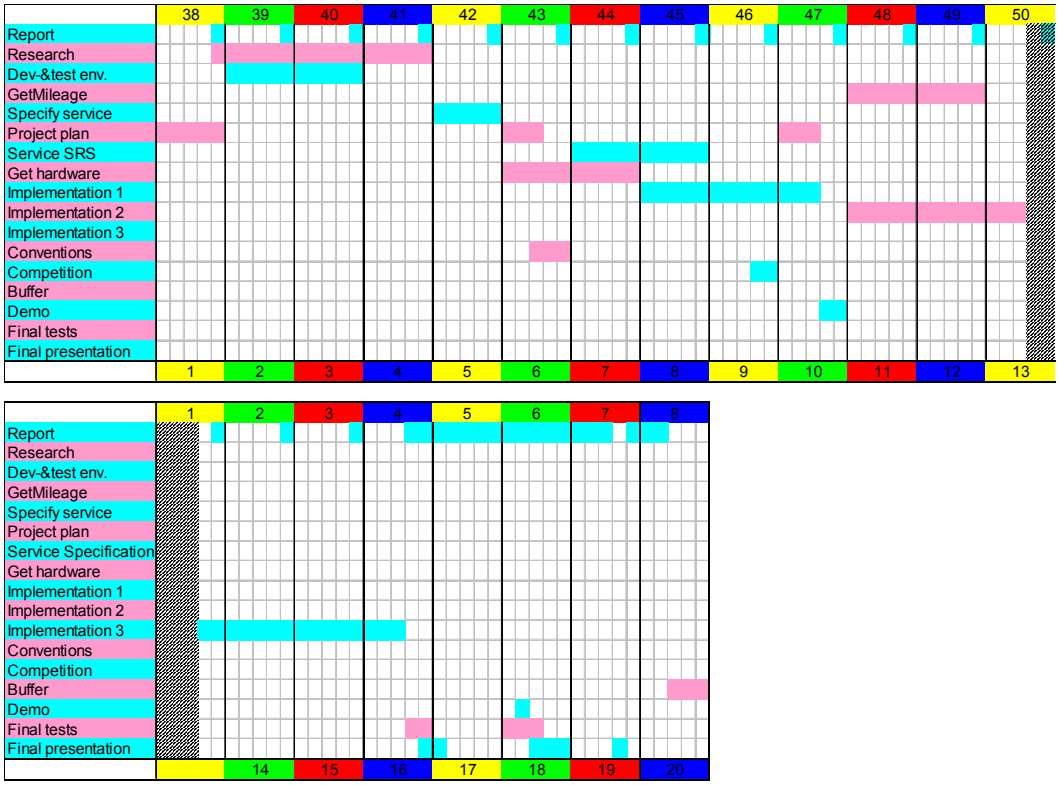


***Figure 5* – Project time plan**

We decided to divide the implementation process into three phases:

- **Phase 1: Basic functionality**
    - Installation of back-end services (MySQL, Axis, etc)
    - Test functions for Simple Object Access Protocol (SOAP) connectivity
    - Test functions for communicating with the diagnostic bus using Java Native Interface (JNI)

- **Phase 2: Diagnostic Trouble Code (DTC) handling**
    - Database structure for vehicle information and DTCs
    - Wireless connection with mobileIP
    - Functionality for receiving, interpreting and storing DTCs at back-end
    - Functionality for handling DTCs at back-office
- **Phase 3: Logging**
    - Database structure for log data
    - Message format and -protocol for transmitting log data
    - Functionality at back-end for requesting, receiving and storing log data
    - Functionality for handling logging at back-office

The solutions were developed in java using Gatespace OSGi frameworks UbiServ and Knopflerfish.

The development process was iterative [WIKIPEDIA], which suited us perfect since we were able to take advantage of what we learned in the early steps when we increased the functionality later on in the process.

Thanks to the work environment at VTEC, the three of us were able to work very closely together. We made use of this advantage a lot and not a day passed by without us talking to each other. Decisions and specifications were made together, which enabled the software development to be split. During development, we relied on the specifications and javadoc interfaces for the implementation, thus freeing one developer from being dependent on the others.

Meetings with our supervisor at VTEC were held whenever needed, often several times a week, which was of great value for the thesis work. This made us less dependent on our supervisor at Chalmers, with whom we only had a few meetings during the entire process.

4. Analysis

4.1. SWOT analysis of GST

By doing a SWOT analysis, it is possible to focus on the strengths and weaknesses in a project and consider the opportunities and threats facing the project [SWOT]. We use this to break down GST and investigate what GST can do for Volvo truck brands, VTEC and vehicle owners. *Table 2* shows our analysis of GST from a general point of view.

Strengths	Weaknesses

• Platform independence
• Modularity
• Standardized:
  -Environment
  -Interfaces
  -Protocols
• Ability to easily add and remove services without changing the hardware
• Collaboration between many different organisations
  -Various ideas
• High degree of freedom to tailor Human Machine Interface (HMI)
• Same hardware can be used for different products
	• Difficult to standardize
	  -Participants have different interests
	  -Participants try to protect their interests against competitors
	• All types of services might not be supported in the standardized environment
	• Difficult to develop services without access to necessary basic GST building blocks/components
	• High demands on hardware and performance - expensive hardware
	• Many similar services can confuse customers
	• No standard for HMI
	  -Different look and feel from different service providers
	  -Varying hardware for user interfaces
	  -Sharing input interfaces between services
	  -Sharing output interfaces between services

Opportunities	**Threats**

• Standardized ready-to-use building blocks can provide support when developing new services
• Platform independence lead to wider market (Develop once – deploy everywhere)
• Possibility to develop services without knowledge of hardware
• Room for small companies
• Increased range of services
• Increased use of telematics
• Increased vehicle safety
• Faster production times and shorter time-to-market.
• Improved infrastructure for telematics services
• Possibility for standardized external connections – off-board services
	• Allows external providers to access the in-vehicle system
	  -Increased competition
	  -Customers might be unaware of who the real provider is
	• Security
	  -Attacks (malicious software, viruses, hackers)
	  -Integrity
	• Increased complexity
	• Services are incompatible with the standards or are harder/ more expensive to adapt to them.
	• The market for telematics is still immature and insecure
	• Risk for increased licence costs
	• Existing services need to be redeveloped
	• All systems must follow the same standard to benefit from the standard

***Table 2* – SWOT analysis**

4.1.1. GST for Volvo truck brands

One of the strengths of the GST architecture is the platform independency. The platform independency makes it possible to use the same hardware for several different applications and therefore use this hardware in different kinds of vehicles within the Volvo group. This could decrease the production cost for both hardware and software due to higher production volumes. On the other hand, the demands on the hardware will be higher. This makes the hardware more expensive since the architecture requires running a Java Virtual Machine (JVM). Today a specific hardware platform is used only for services within Dynafleet. It is possible to add and remove services as long as they exist in the preinstalled Dynafleet software. However, it is not possible to add applications at runtime as it will be in the GST architecture.

GST could lead to increasing competition between car manufacturers since it forces them to open up their telematics platform, both to other manufacturers and to service providers in the telematics market. It can also become harder to defend the car brand, since it might be unclear to the customer what services are provided by the car manufacturer and what services are provided by external suppliers. It is however not very likely that the platform will become as open as GST want, since the manufacturers still will be responsible for the vehicle and therefore must have sufficient control to be able to make guarantees.

4.1.2. GST for VTEC

VTEC does not necessary have to work for companies within AB Volvo, so an open platform creates a large opportunity to sell services on a wider market. There can also be an increased possibility of sharing code and modules between different models and different types of vehicles (cars, trucks, boats) within AB Volvo and Ford Motor Company. The modularity gives an opportunity to lower production times and/or offer more functionality, since each module can be tested independently. The downside is that there will be increased competition from other telematics companies.

4.1.3. GST for the customers

The customer will probably see a broader range of services available for their choice of vehicle when it comes to functionality, quality and price. The risk of lock-in is significantly lower but there is an increased risk of confusion for the customer. The customer needs to take a lot of parameters (such as compatibility and integration between services) into consideration, parameters that traditionally have been taken care of by the car manufacturer. There is also an increased risk of support problems since there no longer is a single provider responsible for all the equipment and services, much like the problems experienced with computer software described in [CS2001].

4.2. Service analysis

4.2.1. Motivation for diagnostics

In the beginning of this project, one task was given: Implement a service using the GST framework. The type of service was not given, but VTEC were interested in a service directed towards the trucking industry, preferably something within the safety area since Volvo is well known for their focus in this area.

We discussed several different services, from multimedia and personal infotainment to goods identification/tracking and network coverage logging/prediction. *Appendix B – Service analysis table* shows a compilation of different services that we considered and their requirements. We categorized the services by their bandwidth requirements. Services where SMS communication is sufficient was categorized as low bandwidth, while those requiring a few tens of kilobits were classified as medium bandwidth. Services requiring more bandwidth are classified as high bandwidth. We also considered which external interfaces the services required i.e. storage, display and keyboard. Finally, we considered external information sources such as diagnostics, positioning and cargo information. Using the table as a starting point, we discussed the different services and their possibilities.

The infotainment services were discarded because they require a lot of expensive hardware, something that is not likely to be pre-installed in Volvo trucks. We think that a user is much more likely to have a personal infotainment center, perhaps a laptop with built-in DVD player or a 3G cellular phone with video and messaging capabilities. The development in this area is very rapid, with new units emerging all the time, which also indicates that the hardware is less likely to be integrated into the vehicle.

The “eCall” and “Rescue”-services were already planned to be developed at the Gothenburg test site for GST, and an earlier thesis work at VTEC has already developed systems for goods identification and tracking. Coverage logging/ prediction was of great interest to us, but was finally discarded, mainly because this kind of service was not very interesting for VTEC [TELECORE].

According to the results of the investigation presented in [TELECORE] the companies in Volvo Group were interested in a RVD service. In order to get more information about which RVD functions the companies wanted the service was divided into three levels:

1. Extract active error codes remotely
2. Run diagnostic query, extract previously active error codes and run in-built test programs
3. Upgrade software remotely

To be able to implement support for level 2 and level 3, specific software for authorization is needed. Due to the problems described in section 4.5.1.1 we were not able to implement any functions that required authorization. The highest level we can reach is therefore 1.5 since we can implement diagnostic query and extract previously error codes but not run in-built test programs.

By doing the thesis work at VTEC, we had an opportunity to access the communication bus inside the vehicle, something that is not always available at other companies. We wanted to take advantage of this opportunity and saw the possibility to create a function-rich remote diagnostics service.

After studying earlier projects involving diagnostics, we visited Volvo Truck Center and discussed our ideas with two service technicians and a Volvo Action Services (VAS) operator. The visit gave us important information about how truck service is handled in the real world (see 4.2.2). They confirmed that advance information about the condition of a vehicle can be very useful and that today's solutions are insufficient. It is possible to retrieve information from vehicles equipped with Dynafleet Online, but this is seldom used because so few vehicles have this functionality.

Therefore, the goal for our service became to provide means for truck centres to lower service times and thereby offer better service to the truck owners. The service is not intended for use directly by the truck owners, nor is it intended to interact with truck drivers. The service could also be used to increase uptime, which is increasingly important for Volvo truck brands since more and more vehicles are leased to customers instead of sold.

4.2.2. Diagnosis workflow at Volvo Truck Centre Bäckebol

If a driver experiences a problem with the vehicle, the driver can call VAS. The driver is asked to describe the problem. The VAS operator (often together with a technician) makes an educated guess of how the problem can be resolved, expected time for repair and what replacement parts are needed. The guess is made solely on the information provided by the driver. A technician with suitable expertise and a timeslot that fits the driver's schedule is chosen. A booking is made and replacement parts are ordered if needed. When the service is booked, a work order is created and assigned to the chosen technician. The work order contains the driver's description of the problem and basic data about the vehicle.

When the driver arrives at the Volvo truck centre, the vehicle is parked outside and is registered at the reception. The key to the vehicle is handed over to the technician together with the work order. The technician drives the vehicle to an empty service bay and starts with performing a diagnostic test of the vehicle. By reading error codes and performing diagnostic tests the source of the problem can be deduced. Sometimes it is also needed to log data while driving. Adjustments and reconfigurations are made and the failing parts are replaced. Occasionally the required parts are not readily available at the service shop and thus have to be delivered from a central warehouse, which introduces additional delays.

4.3. Service scenario

A truck is transporting timber in northern Sweden near Kiruna. The transport's final destination is Luleå and the truck has covered almost half of the distance. It is in the middle of the winter and the snow is falling. The sun has set and it is getting dark. Suddenly the truck's engine starts to run badly and the power of the engine decreases. The truck driver does not know what is wrong with the engine and calls VAS for assistance. The operator starts a service portal, looks up the truck in the database and sees that the truck is equipped with a basic diagnostic service. The operator can now send a request to the truck for a diagnostic report.

The truck can answer the request and send a report either via Short Message Service (SMS) or General Packet Service (GPRS), but right now there is no GPRS coverage available. The truck answers the request via SMS and sends only a minor amount of information, for example the truck's active error codes, due to limitation of SMS size and transmission costs. The operator can now see that there are no error codes set in the truck but according to the driver the truck is still running badly. To be able to get more information about the problem the operator wants to log some engine data. Unfortunately, no logging service is installed in the vehicle. However, the operator is able to order a deployment of the logging service. He defines specific parameters to be logged and for how long time the logging session should last. When GPRS coverage is available, the logging service is downloaded to the vehicle and installed. The logging application starts to log the requested parameters and sends them to the service centre. The driver can continue his journey while the logging application runs. As the data is collected, the operator can examine the result on the web portal using tables and plots and try to single out the source of the problem. If he suspects that a certain part is malfunctioning he can inform the driver about the problem and book a time at a truck service shop in the area where the truck is operating. He can also make sure that a replacement for the suspected malfunctioning part is available at the chosen truck service shop when the truck arrives.

The driver delivers the goods and drives to the truck service shop where the service has been booked. A full diagnostic test can now be made to verify the problem source and the failing part is repaired or replaced. When the service is finished the service engineer reports to the truck owner what is done and the cost for the service. He also informs the driver that the truck is ready for pickup.

4.4. System overview

Our proposed solution enables operators to retrieve diagnostic information from any vehicle supporting the GST platform. In addition to retrieve diagnostic information the system provides information about the vehicles stored in the database such as owner contact information and custom notes.

By giving the operator the possibility to perform thorough diagnostics tests while the vehicle is still operating, time for service can be minimized. The uptime of the vehicle can be increased, thus giving the customer better service. There is no need for interaction with the driver.

The system is composed of three major parts:

- Back-office
  - Provides a simple overview of manageable vehicles
  - Graphical user interface for browsing retrieved vehicle data
  - Enables an operator to retrieve new data from the vehicle
  - Functionality for saving notes on a per vehicle basis
  - Displays owner information for the selected vehicle
- Back-end
  - Provides core functionality for storage and retrieval of data
  - Interpretation and conversion of vehicle data

- On-board system
  - Communicates with the Electronic Control Units (ECUs) on the vehicle communication bus on request
  - Buffers logged data and delivers data to the back-end
  - Manages scheduling of log sessions

The functions of the system can be divided into a DTC request and a log request. For more information about the system see *Appendix A – Service requirements specification*.

4.5. Implementation

4.5.1. Connectivity

Connectivity used in our solution can be divided into two main parts: The connection between the Telematics Control Unit (TCU) and the J1587 diagnostic bus and the Internet Protocol (IP)-based connections. The diagnostic bus is used only inside the vehicle while all external communication is IP-based. *Figure 7* shows an overview of the IP-based network connectivity. A third part, which was not implemented, is SMS capabilities. SMS capabilities is useful when low bandwidth is sufficient and no GPRS coverage is available. Implementing SMS capabilities on the on-board system is only a matter of communication with the cellular phone using AT commands [HAYESAT], which could have been implemented easily. However, in order to successfully communicate using SMS, the back-end also needs SMS capabilities. This can be solved by connecting a cellular phone directly to the server or using a SMS service on Internet, but we did not investigate this further.

4.5.1.1. TCU ⇔ J1587 diagnostics bus

Most trucks have a communication bus called J1708, to which the different ECU:s are connected. J1708 is a standard from Society of Automotive Engineers (SAE). [J1708] defines interface requirements, system protocol, and message format for the bus. The bus is based on a RS-485 serial connection. On top of this, a standard further describing format of messages and data is used. This standard is called J1587 and more information can be found in [J1587]. J1587 is the counterpart to the Controller Area Network (CAN) bus standard used in most cars.

Communication on the diagnostics bus is message-based and uses a set of well-defined addresses called Message Identifications (MIDs) and Parameter Identifications (PIDs). The MID identifies a component connected to the bus (e.g. MID 128=Engine #1) and the PID identifies the data from this component (e.g. PID 084=Road Speed). There are also a third set called Proprietary Parameter Identifications (PPIDs) that require an authorization process.

To be able to communicate on the bus, our TCU had to be equipped with a J1587 interface. This includes both hardware that is able to interface the bus and software that adheres to the message format used on the bus.

Early in the project, we learned that Volvo 3P (V3P) had the necessary hardware and software, but after a while it stood clear that they were not very willing to let us use their software, since it would let us access PPIDs that required authorization, and V3P recommended us to contact Volvo Parts. At a meeting with Carl Johan Andersson at Volvo Parts, we learned that they had a solution tailored to the Pocket PC platform and we got contact information for Jonas Kuschel at the IT-university of Göteborg. Kuschel has been supervising several projects on diagnostics using an iPAQ and though him we got the jacket that provides a J1587-interface for our TCU (see *Figure 6*). Along with the jacket, we got software that is used to communicate with the J1587 bus. This software does not provide any authorization possibilities, which imposes some restrictions on the functionality of our product.

***Figure 6* – The J1587 interface jacket and the TCU**

4.5.1.2. TCU ⇔ Internet

The TCU is connected to another iPAQ (h6340) using a Bluetooth connection. The h6340 has a built-in GPRS modem, which provides Internet connectivity through a service provided by the United Kingdom-based operator Wyless. Wyless provides fixed IP address and roaming capabilities in 80 countries [WYLESS]. This makes it possible for the TCU to keep the same IP address while travelling. We first tried using a Nokia 6310 instead of the h6340, but the connection proved to be very unstable. Our next solution was to run all the TCU software directly on the h6340, but the pin configuration of its I/O-port was different from the h5450, which resulted in incompatibility with the jacket.

Using two PDAs leads to increased instability, but the solution has proved to give sufficient uptime for prototype purposes. If our service was to be implemented in a vehicle, it would most likely run directly on the in-vehicle TCU, which already has connectivity both to the Internet and to the J1587 bus.

4.5.1.3. Back-end

The back-end is connected to Internet through a leased line. This enables the back-office to connect to the back-end. In order to be able to access the on-board system directly, a Virtual Private Network (VPN) tunnel to Wyless is established. The tunnel is Point to Point Tunneling Protocol (PPTP) and Microsoft Point to Point Encryption (MPPE)-based and is using the ordinary Internet connection. The VPN server is located in the UK, which introduces a slight delay, but the delay is very low compared to the access times for the GPRS connection.

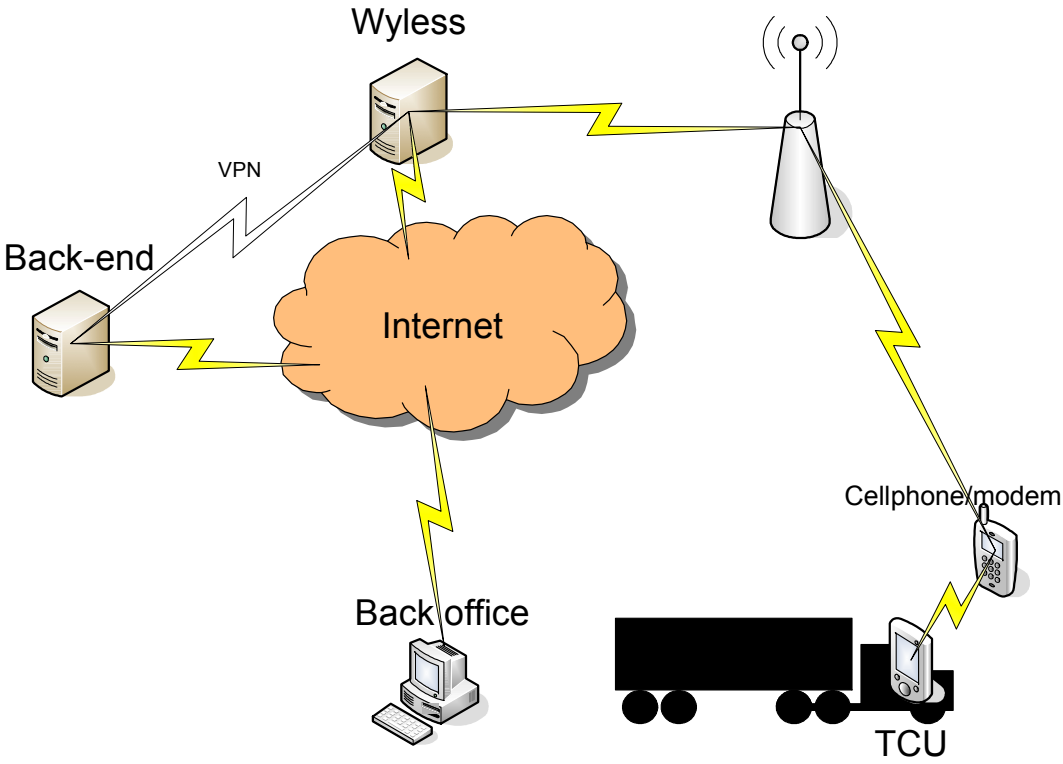


***Figure 7* – IP-based network connectivity overview**

4.5.1.4. Back-office

The back-office requires only basic connectivity over Hyper Text Transfer Protocol (HTTP) to the back-end. This enables the application to work behind many firewalls and through proxies, which is good for corporate environments [MSDNSOAP].

4.5.2. Communication protocols

In order to maintain full functionality while preserving robustness and flexibility, different sets of communication protocols are used for different tasks.

4.5.2.1. Back-office → back-end

All communication between the back-office and the back-end is done using SOAP, HTTP, Transmission Control Protocol (TCP) and IP. By using these protocols, it is possible for the back-office to work through most corporate firewalls and proxies. The downside is that it is not possible to push information to the back-office. Therefore, the back-office polls for information.

4.5.2.2. Back-end → TCU

Our original idea was to use SOAP also on the TCU, but we did not manage to find any working soap clients or servers for our platform. Because of this, and since GST does not specify what protocol(s) to use internally in the service, we decided to implement a server listening on TCP for request. This enables the back-end to connect to the TCU and make requests to log data or probe for DTCs. We developed a text-based message-oriented protocol, which is used for communication. The protocol is partially adapted to suit User Datagram Protocol (UDP) communication, since UDP is less resource-intensive [LAYNET], but full support for UDP has not been developed. Thus, our solution uses TCP only.

4.5.2.3. TCU → back-end

The TCU buffers all data while logging or probing. After retrieving information, the TCU connects to the back-end using a connection very much as the connection described in 4.5.2.2. Content is delivered and the back-end stores the information in a relational database.

4.5.3. Hardware

4.5.3.1. Back-end

Due to firewall problems, VTEC has a few off-site servers. We were provided access to one of these, a HP ProLiant DL380 equipped with dual Intel® Xeon™ CPU 3.06GHz and 512MB of RAM, which is more than enough for our needs.

4.5.3.2. TCU

The in-vehicle system must fulfil certain specifications. The system must have a connection to the vehicle communication bus to be able to extract data from the vehicle control units. In our case, we needed hardware to communicate with the J1708 bus. The in-vehicle system also needs access to Internet through a communication carrier and to fulfil the standards of GST the in-vehicle system also needs an OSGi-framework installed which in turn requires a JVM. All this requires more modifications and upgrades of the hardware used in Volvo trucks today.

To resolve this problem, new hardware is used in the GST project. Our intention was to use this hardware, but delivery of the hardware was delayed so we had to find another solution. Our TCU is a HP iPAQ h5450, which has Wireless LAN, Bluetooth and Infrared communication capabilities. Since the goal is to develop a platform-independent service, it is not that important what hardware we use. The iPAQ was available at VTEC, which enabled a quick resolution of the problem. Even though we strive for platform independency, the capacity of the hardware can become a limitation. Therefore, we decided to use the iPAQ instead of e.g. a laptop, since the iPAQ more closely matches the capacity of the hardware used in the GST project.

Due to the problems described in section 4.5.1.2 we also have an iPAQ h6340 serving as a GPRS-modem. The two iPAQs communicate with each other through Bluetooth. The h5450 is also communicating with the in-vehicle communication bus through a jacket provided by Volvo Parts/Kuschel.

4.5.4. Software

4.5.4.1. In-vehicle software

The Java Programming Language was introduced as a programming language independent of the underlying platform. The programmer can write the code once, and run it on any machine that has a JVM installed. However, there are times when applications must communicate with the operating system directly, e.g. methods that are platform dependent and therefore cannot be a part of the Java platform.

The Pocket PC uses a serial port for communication with the vehicle. There are libraries in Java 2 Micro Edition (J2ME) for serial port communication but the hardware (see section 4.5.3.2) uses a package written in C++. It is necessary to access these methods and variables through the Java application and therefore we must use JNI. A more detailed description of JNI can be found in [JNIBOOK].

JNI is a standard programming interface of the Java Development Kit (JDK) that lets the developer combine Java applications with applications written in other programming languages. Native methods are declared in Java and can be accessed in the same way as standard Java methods.

The developer starts with writing the java code and then compiles it. The result is a class file. With the help of a tool called *javah* that is included in the SDK, a header file is generated from the class file. The header file is then included in the C/C++ project and the native methods are implemented. The native code is then compiled into a shared library and the java program is started. See *Figure 8* for a more detailed description.

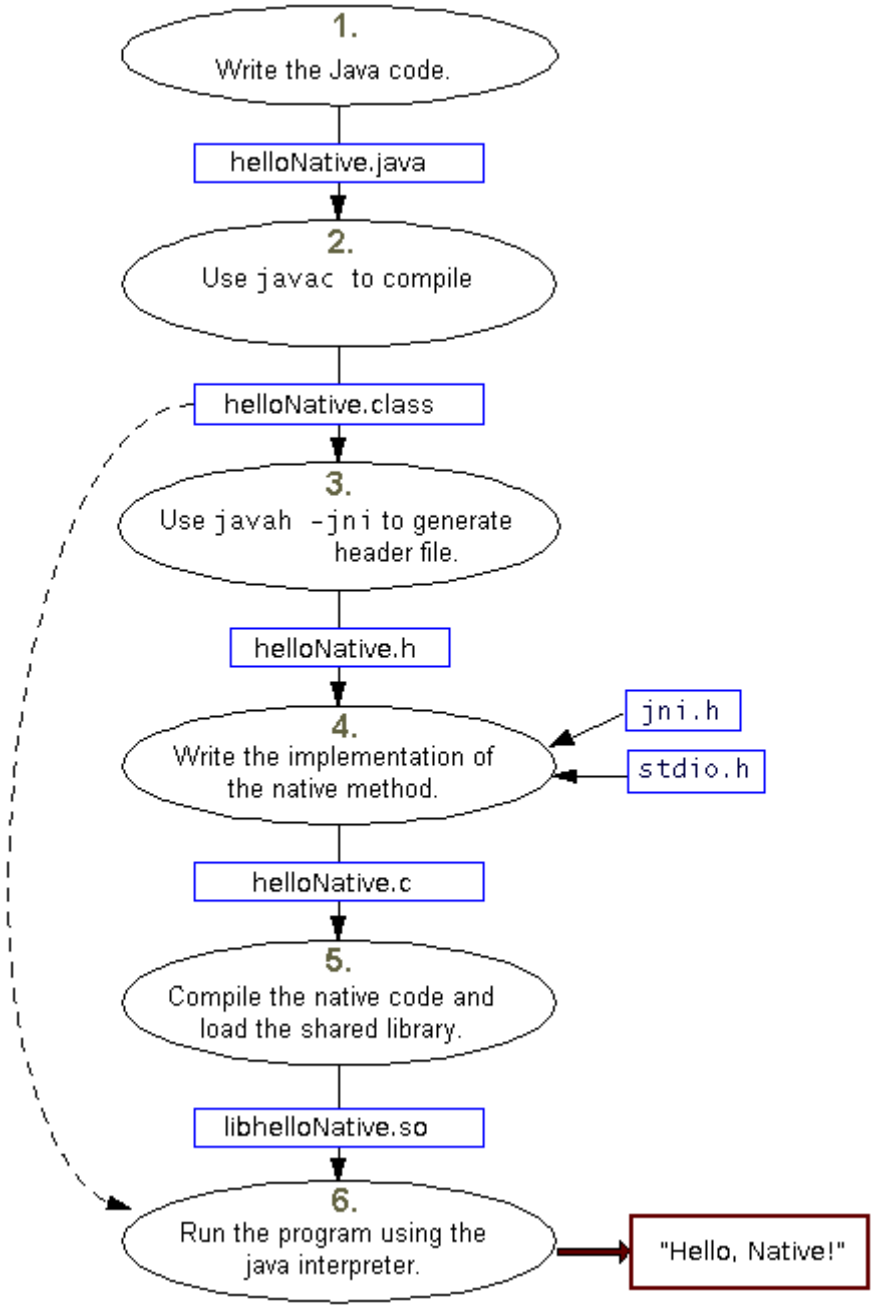


Figure 8 **– Steps for creating an application that uses JNI [JNISTEPS]**

An instance of our NativeObject class is all that is needed from the Java application to communicate with the C++ code. See code snippet below.

```
1  /* Native object for JNI calls */
2  public class NativeObject {
3     public native String send(char[] list);
4
5     static {
6          System.out.println("Java: Native.dll starts loading");
7          System.loadLibrary("Native");
8          System.out.println("Java: Native.dll is loaded");
9     }
10 }
```

Line 3 is just like a regular declaration of a Java method, except for the “native” part. The “native” part tells the JVM that the method will be implemented elsewhere. The Java application must include a static method (line 5-9) that is called by the JVM before any other methods. This static method loads a so-called Dynamic Linked Library (DLL), which is a file that contains executable code that is loaded during runtime. In this case, it contains the methods that were declared in the Java application.

As can be seen in the code snippet, it is possible to pass parameters between the Java application and the DLL file. However, data types are not handled the same way in different programming languages and that must be taken into consideration. JNI provides methods that convert data types in Java to C/C++ and vice versa. Conversion is also possible for other programming languages.

The possibility of combining Java code with code written in another language can be useful. One drawback is that the concept of platform independency is lost. The OSGi Service Gateway provides a way to package DLL files into the bundles. By doing this, it becomes easy to change the bundle that contains the specific platform DLL file with another bundle, that is suitable with the new hardware,

Since the Pocket PC has more limited hardware than a PC, J2ME is used instead of Java 2 Standard Edition (J2SE). Only a subset of the classes that J2SE provides is included in J2ME. AWT (Abstract Windowing Toolkit) must be used instead of the more powerful Swing toolkit, which is a limitation when designing a Graphical User Interface (GUI).

We found it more appropriate to test our application as much as possible on the PC instead of on the Pocket PC for a number of reasons. First, it saved us a lot of time because it is time consuming to transfer bundles to the Pocket PC each time a change in the code has been made. Secondly, from our experiences, the JRE that we used on the Pocket PC is unstable. The Java application could make the JRE crash for no reason and also crash the console, which we often used for debugging.

Figure 9 shows Gatespace’s Knopflerfish running on the PC.

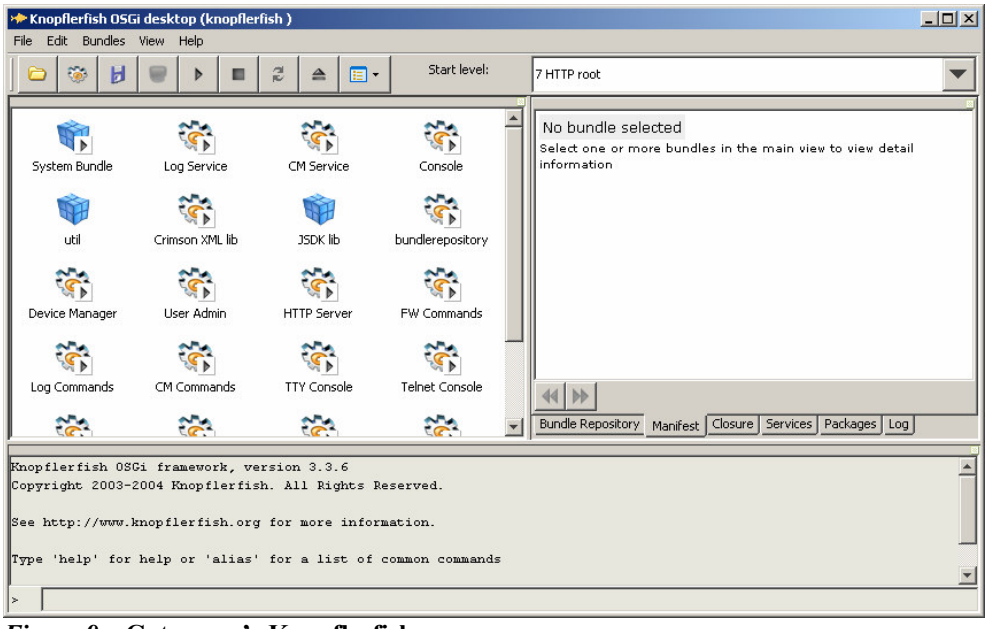


***Figure 9* – Gatespace’s Knopflerfish**

In the framework, it is possible to handle the bundles' different states, which were described in section 2.5.4. *Figure 10* shows a framework running on the Pocket PC, which works almost as the PC version except that a smaller number of bundles are supported.

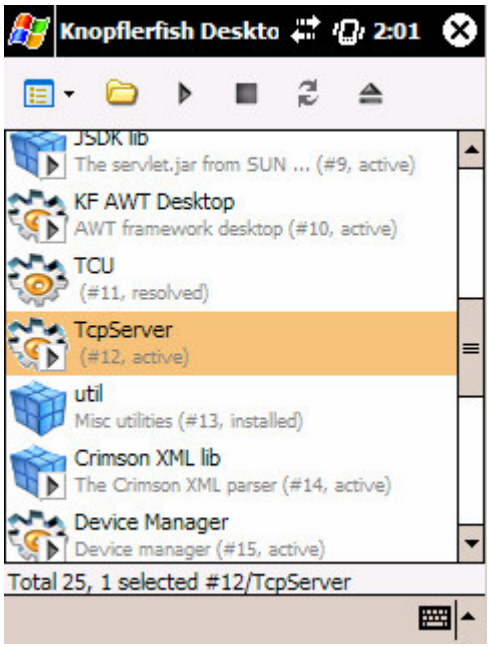


***Figure 10* – The framework running on Pocket PC**

During the development of the in-vehicle system, we worked with the different parts separately, making them work independently of each other. By doing this, it became easier to debug and our application were better structured.

After our first bundle was up and running on the Pocket PC, the next step was to make the C/C++ program that handles the serial port communication work together with the bundle. *Figure 11* shows our test application of all the different parts collaborating.

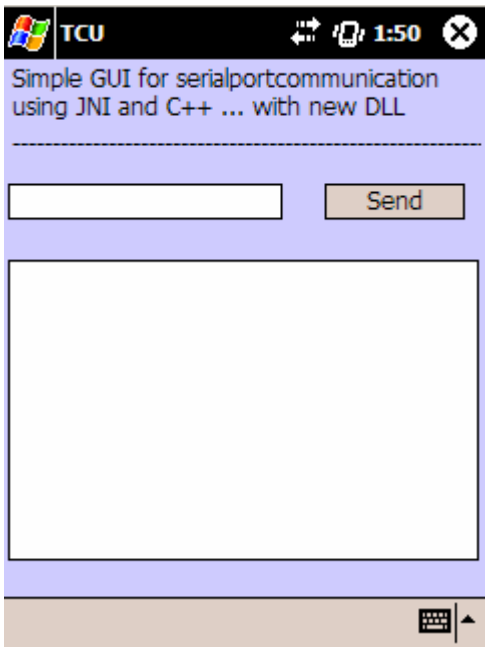


Figure 11 – **Our test application**

With our test application, we could easily test all the different parts of our application to ensure that they all worked together. The Pocket PC is connected to the vehicle and the user writes the input string in the input field and presses the send button. The method that is executed when the user presses the send button is the one that was described in the code snippet above. If everything turned out well, the vehicle will respond with output data presented in the textbox.

Most of the code in the test application was re-used in the real in-vehicle system, which consists of a single bundle that contains both a TCP server and a TCP client for communication with the back-end.

In section 2.5.4 the BundleActivator interface was described. The code snippet of our implementation is presented below.

```
1  /* Sets up and starts the bundle, including the TCP server */
2  public class Activator implements BundleActivator {
3
4     Listener lst;
5
6     public void start(BundleContext bc) {
7           System.out.println("TcpServer bundle starts...");
8           lst = new Listener();
9           lst.start();
10    }
11
12    public void stop(BundleContext bc) {
13          lst.setRunning(false);
14          System.out.println("TcpServer bundle stops...");
15    }
16
17 }
```

The start method creates an instance of our class Listener which is a subclass of the Thread class. It opens up a port and waits for a client to connect. When a client connects, the necessary methods for accessing the DLL are executed and the result is sent back to the back-end with help from the TCP client.

4.5.4.2. Back-office system

Installing an OSGi framework at the back-office is unnecessary complex. One of the reason for this is to minimize the number of downloads and software for the user. Since the life cycle of the back-office system is more static than for the in-vehicle system, a stand-alone java desktop application is all that is needed.

An improvement would be to have the back-office system operate as a web application in the user's web browser, e.g. the user does not have to install any software to the desktop (see 6). An applet is one solution and our application, with only a few adjustments, can also operate as one. We did not take this approach, mainly because the java applet security model requires the user to take additional actions in order to run the applet [JAVAPP]. We wanted to minimize the amount of work for the user as much as possible.

The user is required to have a JRE installed. However, in *Appendix C – Detaljplan förlängning* we give an example of a back-office system that does not need any plug-ins while preserving platform independency. The reason for not choosing the later one is lack of time.

When implementing the in-vehicle system there were some restrictions because J2ME was used. For the desktop application, we had access to several more classes and could use the Swing toolkit for the graphical implementations instead of AWT.

When the program starts, it will try to connect to the server at the back-end. If a connection is made, the program will send the necessary Remote Procedure Calls (RPCs) using SOAP to receive data that will be presented to the user when the GUI appears.

To understand how these SOAP-RPC requests work, a small set of code snippets is necessary and presented below.

```
1  /* Call object for contacting the back-end */
2  private Call getCall() throws Exception {
3
4     String url = SERVER + "GST_diag.jws";
5     Service service = new Service();
6     Call call = (Call) service.createCall();
7     call.setTargetEndpointAddress(new URL(url));
8     return call;
9
10 }
```

The first method getCall is a private method for creating an instance of the Call object. The Call object is used to create and configure the message that will be sent to the server. For more information about this, see [SOAPC].

```
1  /* Fetches timestamps from the back-end */
2  public String[] getVehicleDTCTimestamps(String regno) throws Exception {
3
4     Call call = getCall();
5     call.setOperationName(new QName("", "getVehicleDTCTimestamps"));
6
7     QName stringArr = new QName("", "ArrayOf_xsd_string");
8
10    call.registerTypeMapping(java.lang.String[].class, stringArr, new
11      org.apache.axis.encoding.ser.ArraySerializerFactory(), new
12      org.apache.axis.encoding.ser.ArrayDeserializerFactory());
13
14    call.setReturnType(stringArr);
15
16    call.addParameter("regno", XMLType.XSD_STRING, ParameterMode.IN );
17
18    Object ret = call.invoke(new Object[] {regno});
19
20    if(ret == null) return null;
21
22    if (!(ret instanceof String[])) {
23          System.out.println("Received problem response from server: "+ret);
24          throw new AxisFault("", ret.toString(), null, null);
25     }
26
27    return (String[])ret;
28 }
```

Every SOAP-RPC at the back-office starts by creating an instance of the Call object via the `getCall` method. The `getVehicleDTCTimestamps` method shows an example of this. Most of the methods at the back-office are implemented this way, with the exceptions of different parameter and return types.

Line 5 sets the name of the method to invoke at the back-end. Line 7-14 sets the correct return type. This is a complex procedure, since the return type is an array. On line 16 it is specified how many parameters to pass, and also their names. The invoke method returns an object instance that must be cast to the right type, in this case a string array. Lines 20-25 check if everything went successful.

Now that a basic understanding of the underlying back-office system has been presented, let us have a look at the GUI.

The GUI was inspired by Dynafleet Online, an interface developed to suit the same group of users as our project. It consists of several JPanels [JPANEL] that collaborate with each other. Some of them are updated on regular bases and some are not (see *Table 3*). By designing the GUI this way, it becomes possible to update only certain parts of the application.

Left panel	Only updates when the user presses the update button
Right panel	Updates most frequently
Top panel	Never updates

***Table 3* – Update frequency of panels**

Figure 12 shows the basic structure of the back-office application.

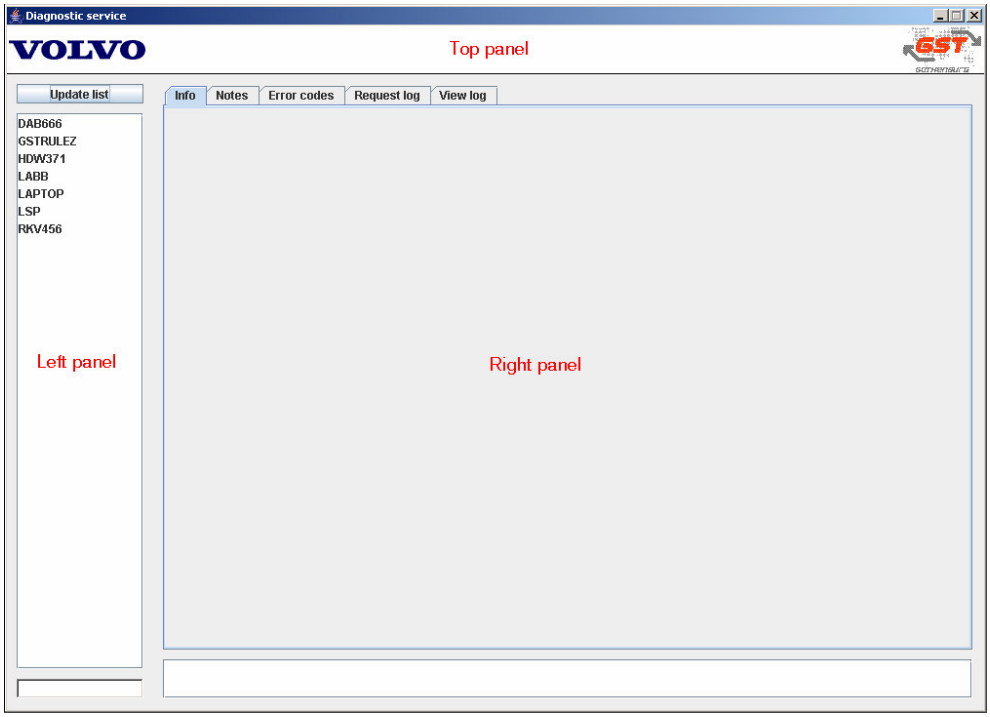


***Figure 12* – GUI structure**

The right panel consists of a JTabbedPane that lets the user switch between a group of components by clicking on a tab with a given title [JTABB]. An explanation of each tab is necessary in order to understand how the back-office works.

The user starts the program and is presented with a list to the left of all the vehicles in the current database. The user can select a vehicle in the list or write the beginning of a vehicle name in the input field, letting the list display only names matching the input.

At the bottom of the right panel there is an output field to which all application components can send messages. This is perfect for debugging but also for letting the user know when events occur, especially since a lot of events occur asynchronously. The first tab that is shown when the user has selected a vehicle from the list is the info tab, which simply displays information about the selected vehicle (see *Figure 13*).

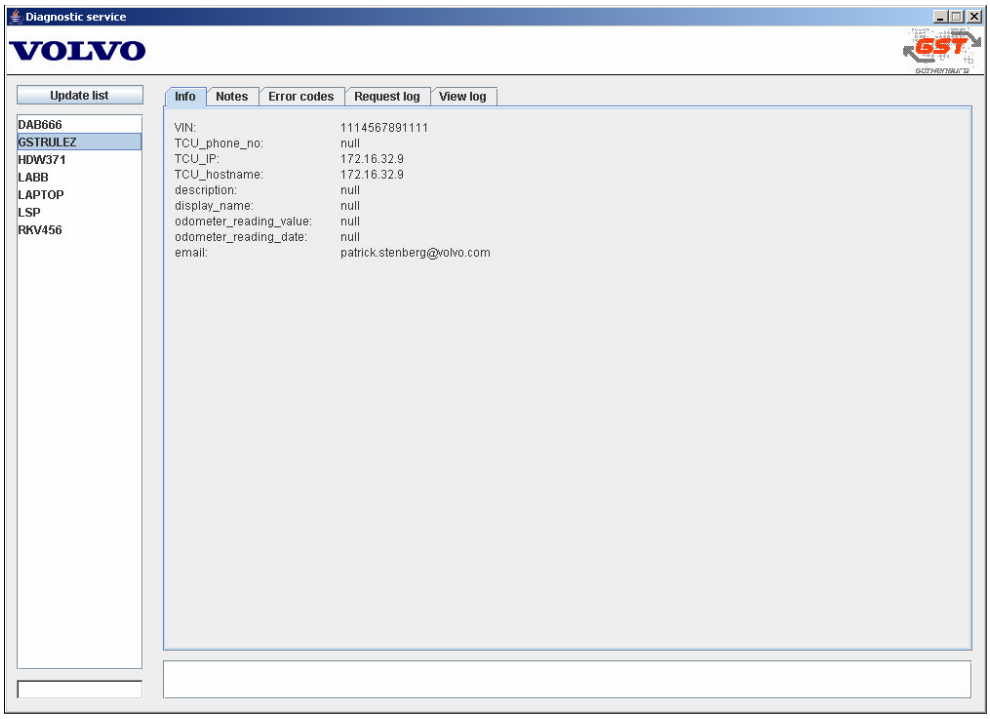


Figure 13 **– Info tab**

It is not possible for the user to change any information in this tab. However, if the user wants to take notes about the vehicle or the driver, there is a notes tab for this purpose (see *Figure 14*).

Notice the save button at the bottom right corner of the notes tab. When the user presses this button, the output field mentioned above, will tell the user if the saving was successful or not.

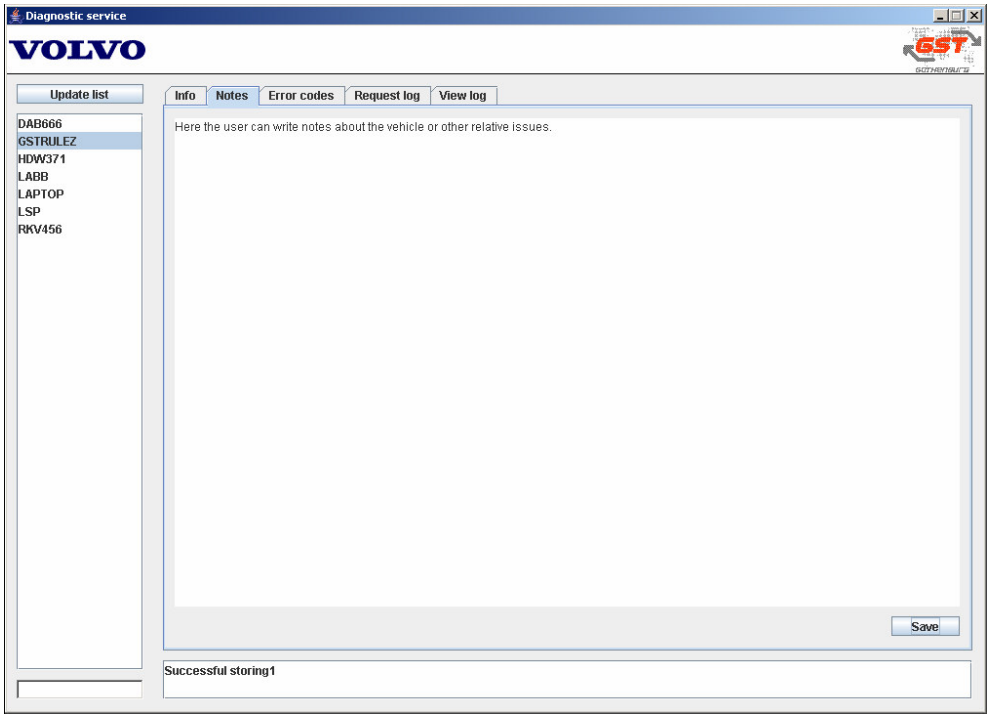


Figure 14 **– Notes tab**

The error codes tab lets the user see all the error codes, filtered by the timestamp of interest, from the selected vehicle. There is a “Request error codes” button that, when pressed, sends a SOAP-RPC to the back-end, asking it to get the latest error codes from the vehicle. The update button simply updates the table with new data from the back-end (see *Figure 15*).

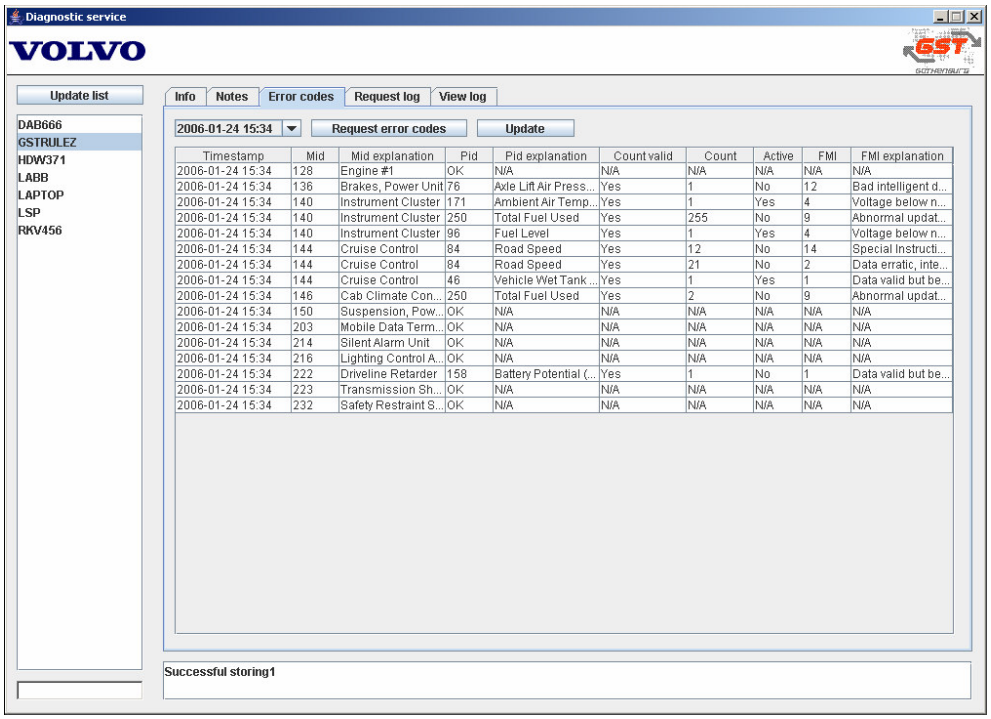


***Figure 15* – Error codes tab**

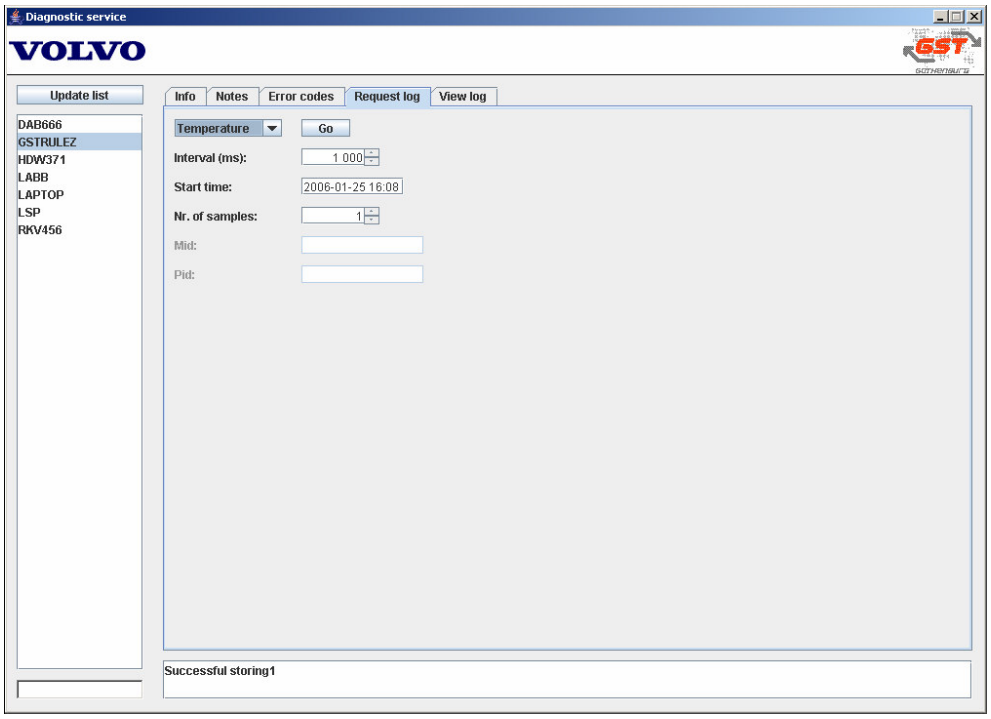


***Figure 16* – Request log tab**

The request log tab lets the user select one of a few predefined log packages from a list and send it to the on-board system. The predefined log packages are:

- Temperature
- Engine data
- Gas pedal
- Special (consists of all the above)
- Rigg

It is also possible for the user to create an own log package but that requires more knowledge about J1587.

The fields for interval and number of samples let the user select the time between each sample and how many samples to acquire. The user can also choose when the logging should start, which enables delayed logging (see *Figure 16*).

The view log tab provides the user with information about the log packages. Thanks to a very nice library called JFreeChart [JFREEC] it became very easy to visualize the results in diagrams (see *Figure 17*).

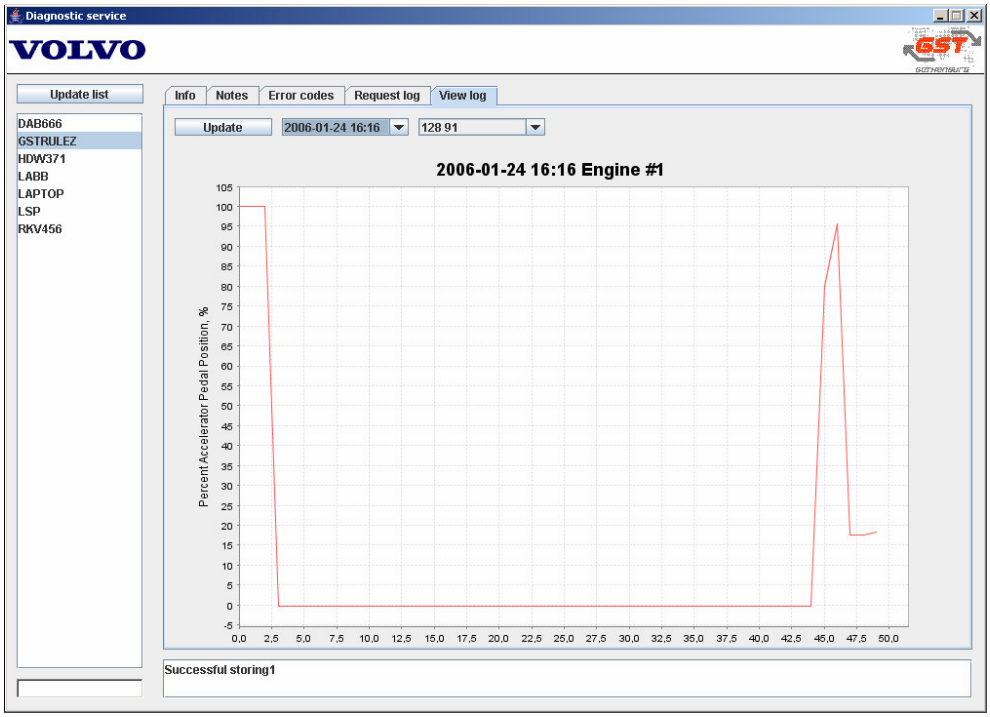


***Figure 17* – View log tab**

4.5.4.3. Back-end

As can be seen in *Table 4*, a variety of software packages are used at the back-end. All software is developed by third parties except Gst_diag and BackendTCPServer, which were developed by us in order to serve as back-end functionality for the diagnostics service.

Software	Information
Webmin	Web-based interface for system administration. [WEBMIN]
Apache httpd	Open-source HTTP server [HTTPD]
Apache Tomcat	Servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies in Java 2 Enterprise Edition (J2EE). [TOMCAT]
Apache Axis	Java platform for creating and deploying web services applications. [AXIS]
MySQL	Open source relational database. [MYSQL]
mysql-connector	Native Java driver that converts JDBC (Java Database Connectivity) calls into the network protocol used by the MySQL database. [CONN]
Java activation framework	Used for helping Apache Axis identify the type of an arbitrary piece of data. [JAF]
Client for Microsoft PPTP	Used for connecting to Wyless, enabling direct interaction with the TCU. [PPTP]
BackendTCPServer	*A service to which TCUs can connect and deliver data.*
Gst_diag	*The core for back-end functionality.*

***Table 4* – Installed software at the back-end**

In the beginning of the project, administration of the back-end server from Volvo was impossible due to firewall restrictions and Volvo IT policy. We managed to set up a Webmin installation, which allowed limited administration over http.

Since GST specifies that communications should be done using SOAP and the available server already had a working installation of the Apache web server and Tomcat, we decided to use Apache Axis as our communication platform. Apache Axis is a java platform for creating and deploying web services applications [AXIS]. Thanks to Axis, our user interface can communicate with the back-end using standardized SOAP-RPC. SOAP-RPC is similar to XML-RPC; both provide means to call external procedures, but SOAP-RPC extends the functionality with support for user defined data types (and more). For an excellent comparison of SOAP-RPC and XML-RPC, see [RPC].

We decided to use a database for simple and safe long-term storage, and the choice of database system fell on MySQL, since it is freely available and easily managed through Webmin. By using the mysql-connector java package, accessing the database from our web services became a trivial matter, as we are able to run SQL queries directly from java. Earlier experiences show that this method is easy to use .

Gst_diag is the core for our back-end functionality. It consists of a set of RPCs. Some calls are available publicly and some are private. The public calls are the only interface to the back-office, and provides all necessary functionality. A typical call, which returns a thorough description of logged DTCs given a vehicle registration number and a timestamp, is shown below:

```
/**
 * Request a thorough description of DTCs
 * @return a list of strings, where each string contains information from a DTC
 *  Example:
 *  2005-12-12 15:01|163|Vehicle Security|224|Immobilizer Security
      Code|Yes|1|Yes|12|Bad intelligent device or component
 * @param regno the license number of the vehicle for which the DTCs are
      requested.
 * @param timestamp a timestamp that should match the DTCs or null if all
      timestamps should be returned
 * @see getDelimiter()
 * @see getDTCDescriptions()
 */

public String[] getVehicleDTCsLong (String regno, String timestamp) throws
      AxisFault, SQLException {
   Statement stmt = initDB();
   if (stmt==null) throw new AxisFault("Failed to connect to database.");
   String ts_filter = new String();
   if(timestamp==null) ts_filter = "";
   else ts_filter = "AND timestamp='"+timestamp + "' ";

   String vin = regnoToVIN(regno);
   if(vin == null) return null;
   ResultSet rs = stmt.executeQuery("SELECT * FROM " + DB_DTCs + " WHERE VIN= '" +
      vin +"' "+ ts_filter + "ORDER BY timestamp DESC");

   Vector DTCs = new Vector();
   if(!rs.next())return null; // Return null if no DTCs for the current vehicle
   fetchDTC(DTCs, rs);
   while(rs.next()) fetchDTC(DTCs, rs);
   return vectorToStringList(DTCs);
}
```

The use of delimited strings results in a format that is quite easy to handle in the back-office software. Another approach could be to define our own XML schemas to be able to use more appropriate data types and –structures, but the solution with delimited strings is sufficient for the functionality we need.

The BackendTCPServer is a TCP-based service that provides means for the TCUs to deliver and store data. It uses a very simple text-based protocol for handshaking and data delivery. Our original plan was to use Gst_diag for this functionality, and let the TCUs connect using SOAP, but since we were unable to find a working SOAP client for the Pocket PC platform, other means of delivery was required. SOAP would have given us a more straightforward solution, but SOAP might not be the best solution since it is not very suitable for small messages and two-way communication. Using a proprietary format over TCP or UDP that is adapted to the specific needs of a diagnostics service could result in more efficient use of bandwidth and other resources such as processing power and memory.

4.6. Service Submission Contest

“To demonstrate the true openness of the innovative architecture developed by the EU-supported GST Integrated Project, this Service Submission Contest (SSC) is published as a competitive call open to all potential Europe-based telematics service providers (public, private and individual) that are ready to develop GST-compliant services.” [GSTWWW]

The total price sum of the contest is € 22,500 for first, second and third place. The contest consists of three phases. The first is called “application phase”, in which the contestants are able to send in applications describing their service. Contestants allowed to continue to phase two are given time to develop their solutions and at the end of this phase the GST-compliance of the services is examined. The GST-compliant services are then considered finalists, out of which the winners will be chosen by a jury. In phase three, some of the services will be demonstrated and tested at the test sites. The winners will be announced at the ITS World Congress in London in October 2006.

In December 2005, we decided that our service was interesting enough to submit to the contest. We applied, and on the 7th February 2006 we informed that our service had been allowed to enter the second phase.

5. Conclusions and recommendations

Based on our research, we decided that a RVD service would be suitable for implementation in this project. The development companies within AB Volvo have great interest in diagnostics and a diagnostic service gave us a challenge to implement since it is very hardware-dependent. Due to problems with acquiring software that enables authorization when communicating with the vehicle diagnostic bus, we had to restrict the functionality to parameters that do not need authorization.

We have showed that it is possible to develop a service in a very short time: after a three-week implementation phase where basic functionality was developed, we were able to implement DTC functionality in only three additional weeks and during the next three weeks we added logging functionality.

We did not manage to make the service fully GST-compliant, mostly because the release of the reference implementations was delayed. The GST platform is very complex and quite hard to comprehend even with full access to all GST documentation, which made it hard to develop an understanding of the standards. We were however successful in making the service OSGi compliant, which should provide a good base both for further work and adaptation to the GST platform.

The service has some platform dependencies since it needs to communicate with the in-vehicle diagnostics bus that requires vehicle-specific hardware, but this dependency could be handled by separating the dependent parts from the diagnostics service. The service could then use a well-defined, standardized interface to another bundle, which abstracts the hardware from the service.

Our investigations show that GPRS is a well-suited carrier for this type of service, both for DTCs and logging, but SMS is enough for retrieving DTCs. The solution from Wyless provides a flexible and reliable connection for IP-based connectivity.

Because of security issues, it is hard to use a java applet or web application at the back-office and our solution is therefore based on a desktop application. This has some drawbacks, e.g. the user has to download and install the application in order to use it. Therefore, an applet or web application is recommended if the security issues can be resolved.

The modularity of GST gives an opportunity to lower production times and/or offer more functionality, since each module can be tested independently. This can be useful for both VTEC and the other development companies within the Volvo group. The downside is that there is a risk for increased competition but there is also a possibility for VTEC to broaden the market and develop for other OEM:s.

We contributed to the GST service submission contest and were granted permission to enter phase 2, which will proceed until June 15th 2006.

6. Future work

VTEC is interested in adding functionality to our solution and has proposed an extension of the thesis work as an in-house development project. Our recommendations for this project can be found in *Appendix C – Detaljplan förlängning*, together with estimations of how much time is needed. The recommendations include further adaptation to the GST platform, some rework of the current implementation and addition of functionality such as real-time and selective logging, remote control of vehicle and integration with one of the telematics units used in today's or tomorrows vehicles.

7. References

[AXIS] ws.apache.org/axis/ 2005-12-06

[CONN] www.mysql.com/products/connector/j/ 2005-12-06

[CS2001] Så tvingar du programleverantören att ta reson
Computer Sweden 2001-12-19
Martin Wallström
Sida: 16

[EUFP6] European Commission FP6
http://europa.eu.int/comm/research/fp6 2006-02-03

[GSTEFCD] QUA_EFCD_Poster.ppt. Michael Ortgiese, PTV AG

[GSTPRE] PRE_GST_IP_AND_OS.ppt, Erwin Vermassen
Available at:
http://www.gstproject.org/os/index.php?content=gstctdocuments

[GSTQTY] GST Quality Plan version 1.7, 2005-09-15
Peter Van der Perre, Helge Sturesson, Erwin Vermassen, Sophie Dupuis (ERTICO), Volker Vierroth (T-Systems).
Deliverable number: DEL_GST_1_1

[GSTRSQ] GST_RSQ_General_Presentation.ppt. Peter Van der Perre, ERTICO

[GSTSAF] Safety Channel Realize Safer Roads!
PRE_GST_SAFCHAN_1_Overview_v1.ppt, Erwin Vermassen, ERTICO
Presentation at GST Kick-Off 6th of April 2004 in Brussels, Belgium

[GSTWWW] Global System for telematics
www.gstforum.org 2005-11-22

[HAYESAT] HAYES AT Commands
http://www.cellular.co.za/hayesat.htm#SMS%20Command%20Set
2006-02-03

[HTTPD] http://httpd.apache.org/ 2005-12-06

[J1587] Electronic Data Interchange Between Microcomputer Systems in Heavy-Duty Vehicle Applications, issues by Truck And Bus Low Speed Communication Network Subcommittee at Society of Automotive Engineers

[J1708] Serial Data Communications Between Microcomputer Systems in Heavy-Duty Vehicle Applications, issued by Truck And Bus Low Speed Communication Network Subcommittee at Society of Automotive Engineers

[JAF] java.sun.com/products/javabeans/glasgow/jaf.html 2005-12-06

[JAVAPP] Applet security
http://java.sun.com/sfaq/ 2006-02-02

[JFREEC] JFreeChart
http://www.jfree.org/jfreechart 2006-02-02

[JNIBOOK] The Java Native Interface - Programmer's Guide and Specification - Sheng Liang - 1997

[JNISTEPS] http://public.cabit.wpcarey.asu.edu/janjua/java/jni 2006-02-01

[JPANEL] Class JPanel
http://java.sun.com/j2se/1.3/docs/api/javax/swing/JPanel.html
2006-02-02

[JTABB] Class JTabbedPane
http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JTabbedPane.html
2006-02-02

[LAYNET] Comparative analysis - TCP – UDP
http://www.laynetworks.com/Comparative%20analysis_TCP%20Vs%20UDP.htm 2006-02-06

[MSDNSOAP] SOAP: The Simple Object Access Protocol -- MIND January 2000
http://www.microsoft.com/mind/0100/soap/soap.asp 2006-02-06

[MYSQL] www.mysql.com 2005-12-06

[ONSTAR] http://www.onstar.com, 2006-01-31

[OSGI] http://www.osgi.org/osgi_technology/index.asp?section=2 2006-02-01

[PPTP] pptpclient.sourceforge.net 2005-12-06

[PROSYST] ProSyst partner projects
http://www.prosyst.com/partners/partner_projects.html 2006-02-10

[RPC] weblog.masukomi.org/writings/xml-rpc_vs_soap.htm 2005-12-06

[SCANIA] Scania Fleet Management
http://www.scania.com/services/fleetmanagement/ 2006-02-10

[SOAPC] Anatomy of a SOAP client
http://www.developer.com/open/article.php/728081 2006-02-02

[SWOT] SWOT Analysis - Management Training from Mind Tools
http://www.mindtools.com/pages/article/newTMC_05.htm 2005-11-22

[TELECORE] Telecore, internt dokument. Greger Landén, Johan Sahlström. Volvo technology 2002

[TOMCAT] tomcat.apache.org 2005-12-06

[VOLVOWEB] Volvo Trucks Global - Transport information systems
http://www.volvo.com/trucks/global/en-gb/services/tis/ 2006-02-10

[WEBMIN] www.webmin.com 2005-12-06

[WEBSPH] WebSphere Studio Device Developer 4.0, IBM 2002

[WIKIPEDIA] Definition of iterative development
http://en.wikipedia.org/wiki/Iterative_development 2006-02-03

[WYLESS] Wyless Features and GPRS Roaming list September 2005, Wyless Scandinavia AB

8. Literature

Here is a collection of documents and articles for those interested in further reading.

Information about the High Level Architecture in GST:
IP High_Level_Architecture (DEL 3.1) - GST High Level Architecture
Peter Van der Perre, Ertico
Avaliable at http://www.gstforum.org/en/downloads/deliverables.htm

Information about the GST subproject Open Systems:
OS_Architecture_and_interface_specifications (DEL3.1) - GST Open Systems Architecture and Interface Specifications
Erwin Vermassen, Ertico
Avaliable at http://www.gstforum.org/en/downloads/deliverables.htm

Emerging telematics services and companies are described in:
Mobilen avslöjar var du befinner dig
Göteborgs-Posten 2005-07-27 Sida: 42
Kenny Genborg

9. Appendix A – Service requirements specification

VOLVO

Företagsnamn/**Company name**	Dokumentnamn/**Name of document**		Sida/**Page**
Volvo Technology Corporation	SRS - Remote Diagnostic Service		1 (17)
Utfärdare (avd nr, namn, tfn, geografisk placering, sign)/Issuer (dept, name, phone, sign)	**Datum/Date**	Bilaga/**Appendix**	Reg.nr/**Reg. No.**
06600, Mikael Johansson, Patrick Stenberg, Robert Laxing 031-669433, M1:6	2005/12/13		
Ärende/**Subject**			Utgåva nr/**Issue No.**
System Requirement Specification - Remote Diagnostic Service for GST			1.16

SRS - Remote Diagnostic Service

System Requirement Specification - Remote Diagnostic Service for GST

Issue 1.16

VOLVO

Volvo Technology Corporation

1 INTRODUCTION 5

1.1 PURPOSE 5
1.2 INTENDED AUDIENCE 5

2 REFERENCES 6

3 GLOSSARY 7

3.1 ABBREVIATIONS 7
3.2 DEFINITIONS 7

4 OVERALL DESCRIPTION 8

4.1.1 Product perspective 8
4.1.2 Product functions 8
4.1.3 Structure and information flow of the system 9
4.1.3.1 Interaction during DTC request service 9
4.1.3.2 Interaction during log service 10
4.1.4 Users of the system 11
4.1.4.1 Primary user 11
4.1.4.2 Optional and possible future users 12

5 EXTERNAL INTERFACE REQUIREMENTS 13

5.1 TCU USER INTERFACE 13
5.2 HARDWARE INTERFACE 13
5.2.1 Handheld computer 13
5.2.2 Connection to vehicle network 13
5.3 COMMUNICATION INTERFACE 13
5.3.1 No contact with server. 13

6 SYSTEM FEATURE REQUIREMENTS 14

6.1 BACK-OFFICE 14
6.1.1 Vehicle selection 14
6.1.2 Information view 14
6.1.3 Notes view 14
6.1.4 Error codes view 15
6.1.5 View log view 15
6.1.6 Request log view 15
6.2 BACK-END 15
6.2.1 Vehicle database 15
6.2.2 Listing of vehicles 15
6.2.3 Vehicle information 16
6.2.4 List stored DTCs for a vehicle 16
6.2.5 Request DTCs 16
6.2.6 List stored logs 16
6.2.7 Get log data 16

6.2.8 Get notes 16
6.2.9 Store notes 16
6.2.10 Run log package 16
6.2.11 Store logging data 16
6.2.12 Store DTCs 16
6.2.13 Optional - Deploy logging service/bundle 16
6.2.14 Optional - Set/View compression settings 16
6.3 TCU 17
6.3.1 Upload error codes 17
6.3.2 Run logging service 17
6.3.3 Upload logging data 17
6.3.4 Optional - Receive log package request 17
6.3.5 Optional – Cancel logging request 17

Table of figures

Figure 1: Diagnostic service information flow 9
Figure 2: Sequential diagram of request of DTC from vehicle 10
Figure 3: Sequential diagram of request of logs from vehicle 11

1 Introduction

1.1 Purpose

Volvo Technology Corporation is among other partners involved in an EU-funded project called GST Global Systems for Telematics. The goal for GST is to create an open environment in which telematics services can be developed and delivered in a cost-effective way [GST]. This will hopefully increase the range of telematic services that will be available for manufacturers and customers.

In order to evaluate what GST could mean for Volvo Technology Corporation a master thesis work, “Evaluating GST using wireless off-board diagnostics”, was formed. The main goal is to make an analysis of what GST together with the OSGi framework can do to decrease the development time for new services. To be able to make a solid evaluation, one of the main tasks with the master thesis work is to develop and provide a competitive service for GST. This is the System Requirement Specification (SRS) for that service.

The introduction of ECU:s (Electrical Control Units) in the car industry in the middle of the nineties has led to a different and more computerized way to repair a modern car. Nowadays a computer is connected to the car and through a diagnostic test the car is asked what kind of problems it has. The diagnostic test is the first action taken at a Volvo repair shop in order to find both known and unknown problems. For Volvo Truck Corporation (VTC) the tool used to make a diagnostic test of a truck is called VCADS Pro (Vehicle Computer Aided Diagnostic System) [EWOD].

To increase the up-time of a truck a lot of work is done considering Remote Vehicle Diagnostics (RVD). If it is possible to make a remote diagnostic, a lot of knowledge about the problem of the vehicle can extracted already before the vehicle reaches the repair shop. This could decrease the amount of time to repair the vehicle making the repair shop more efficient and the customers more satisfied with the service.

The system specified in this SRS is a remote diagnostic service. The unique feature compared to other remote diagnostic services is that it should follow the architecture specified by GST. This document will include all the requirements of the service.

1.2 Intended audience

This document is written primarily for technical people who have at least rudimentary knowledge about software development. We assume basic familiarity with the concepts of software development processes but also provide some basic conventions in this introduction. Some knowledge about the GST project is useful in order to fully understand the service.

2 References

[GST] www.gstforum.org

[EWOD] Evaluating GST using wireless offboard diagnostics, Master Thesis Report…

3 Glossary

3.1 Abbreviations

DNS	Domain Name System
DTC	Diagnostic Trouble Code
ECU	Electrical Control Unit
GST	Global System for Telematics
GUI	Graphical User Interface
RVD	Remote Vehicle Diagnostics
SRS	System Requirement Specification
TCU	Telematic Control Unit
VCADS	Vehicle Computer Aided Diagnostic System
VIN	Vehicle Identification Number
VTC	Volvo Truck Corporation

3.2 Definitions

Up-time	The time a vehicle is working and operating
Down-time	The time a vehicle is in a non-operational state.

4 Overall description

4.1.1 Product perspective

The concept of RVD is a common area of today's research within the car industry, especially among the truck manufacturers. The goal is to increase the up-time of a vehicle. If the mechanics at a repair shop have the possibility to get more information about a vehicle that requires service, before it actually reaches the repair shop, they are able to prepare the service and decrease service time.

The remote diagnostic service is thus no new service. A lot of projects has already designed and implemented remote diagnostics services. The innovation about this remote diagnostic service is that it should follow the standards of the GST project. The adaptation to the GST standards will lead to two major news; platform independency and a new deployment infrastructure of the service.

A common procedure of booking a service is described in the report Evaluating GST using wireless offboard diagnostics [EWOD]. One important observation is that the service centre operator that handles the calls from drivers usually does not have a mechanical education. The operator has to consult a mechanic to make a guess about what the problem with the vehicle could be caused of. The guess is based on the phone call with the driver.

If the operator has the possibility to get more information about the vehicle, e.g. error codes and parameter logs, the guess can be more precise and the consulting of a mechanic can be based on more facts. The information from the vehicle should be easily accessible, but also easy to understand. The interface between the operator and the system should not be more difficult than a search on the internet to minimize the learning time of the system. It must be possible to print logs and error codes to be able to consult the mechanics in the repair shop without having to bring a computer.

4.1.2 Product functions

The following list presents a high-level overview of the functions that will be provided by the system:

- Get information about a vehicle through a webportal.
- Read Diagnostic Trouble codes from a vehicle on a webportal.
- Log parameters from a vehicle.
- View logged parameters in graphs.
- Ability to make notes about a certain vehicle
- Possibility to see job cards and service history about a vehicle. (Optional)
- Possibility to export log data to external formats. (Optional)

4.1.3 Structure and information flow of the system

The remote diagnostic service must adapt to the GST standards regarding interaction and deployment which leads to a certain structure of the service components. GST specifies four systems interacting with each other; the service centre, the control centre, the payment & billing centre and the client. When developing a service the client system and the service centre system must be implemented. In this remote diagnostic service the service centre will consist of the web portal and the back-end. The web portal is the user interface against the operator at the call centre and the back-end handles all the communication with the vehicle and stores the information in a database. In *figure 1* the information flow is described.

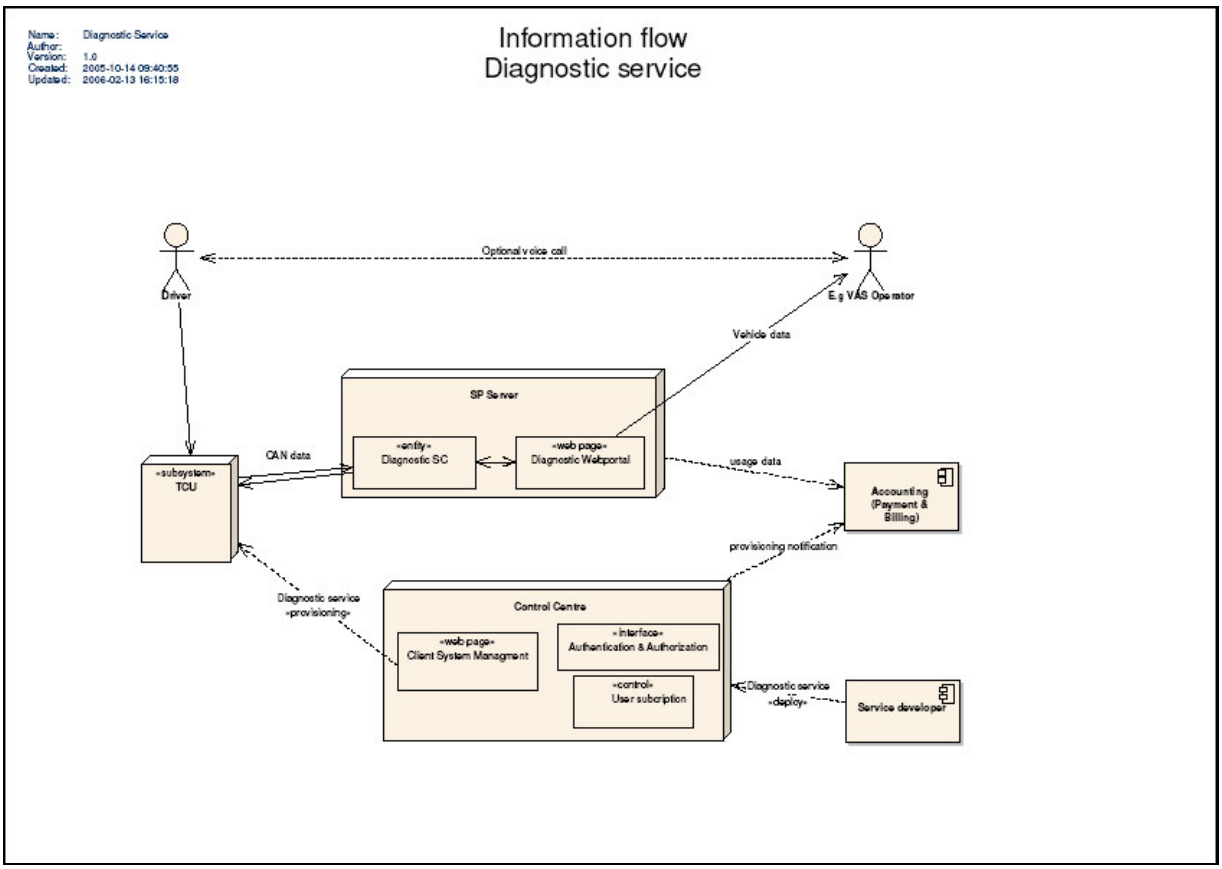


Figure 1: Diagnostic service information flow

4.1.3.1 Interaction during DTC request service

The interaction during a DTC request is explained in the sequential diagram in *figure 2*. The driver experiences problems with the vehicle and calls the Volvo Action Service (VAS) operator. The operator starts the service application and looks up the vehicle's license number in the vehicle list. The information about the vehicle is requested from the back-end's database and displayed to the operator. To get information about the vehicle's DTCs the operator enters the DTC tab in the application and presses the

Request DTCs button. The application sends a request to the back-end and gets the vehicle's old DTCs stored in the back-end's database with timestamps. Simultaneously a request for DTCs is sent to the vehicle via SMS. When the vehicle's Telematic Control Unit (TCU) receives the request it send a request to the Electronic Control Units (ECU) and stores the response. When the TCU has gotten the DTCs it sends them to the back-end via SMS or GPRS. The DTCs are stored in the back-end´s database and also sent to the application. The operator can now discuss what the problem might be with the driver.

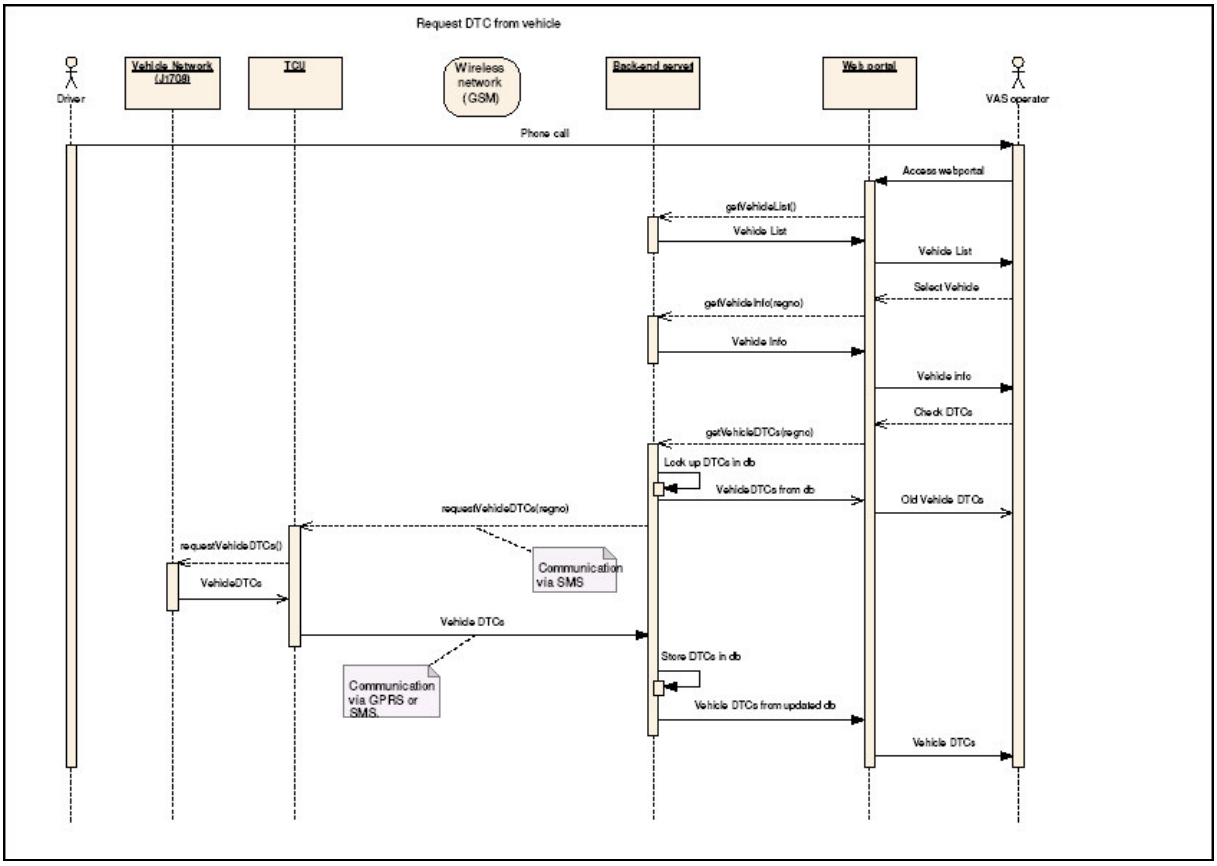


Figure 2: Sequential diagram of request of DTC from vehicle

4.1.3.2 Interaction during log service

If the operator can not get hold of the problem of the vehicle through the DTCs or more information about a known problem is desired a log service can be started. The operator can get a list of all the logs already made on the vehicle and the log data is presented in a diagram. If the specific log that the operator wants does not exist in the list he/she can chose among a set of different log packages that can be sent to the vehicle. When the operator has chosen a log packet the start time and the time between two samples must be specified. Also the number of samples must be specified. When the information is specified the operator presses the request log button and the request is sent to the back-end. The back-end forwards the request to the vehicle via SMS or GPRS. When the

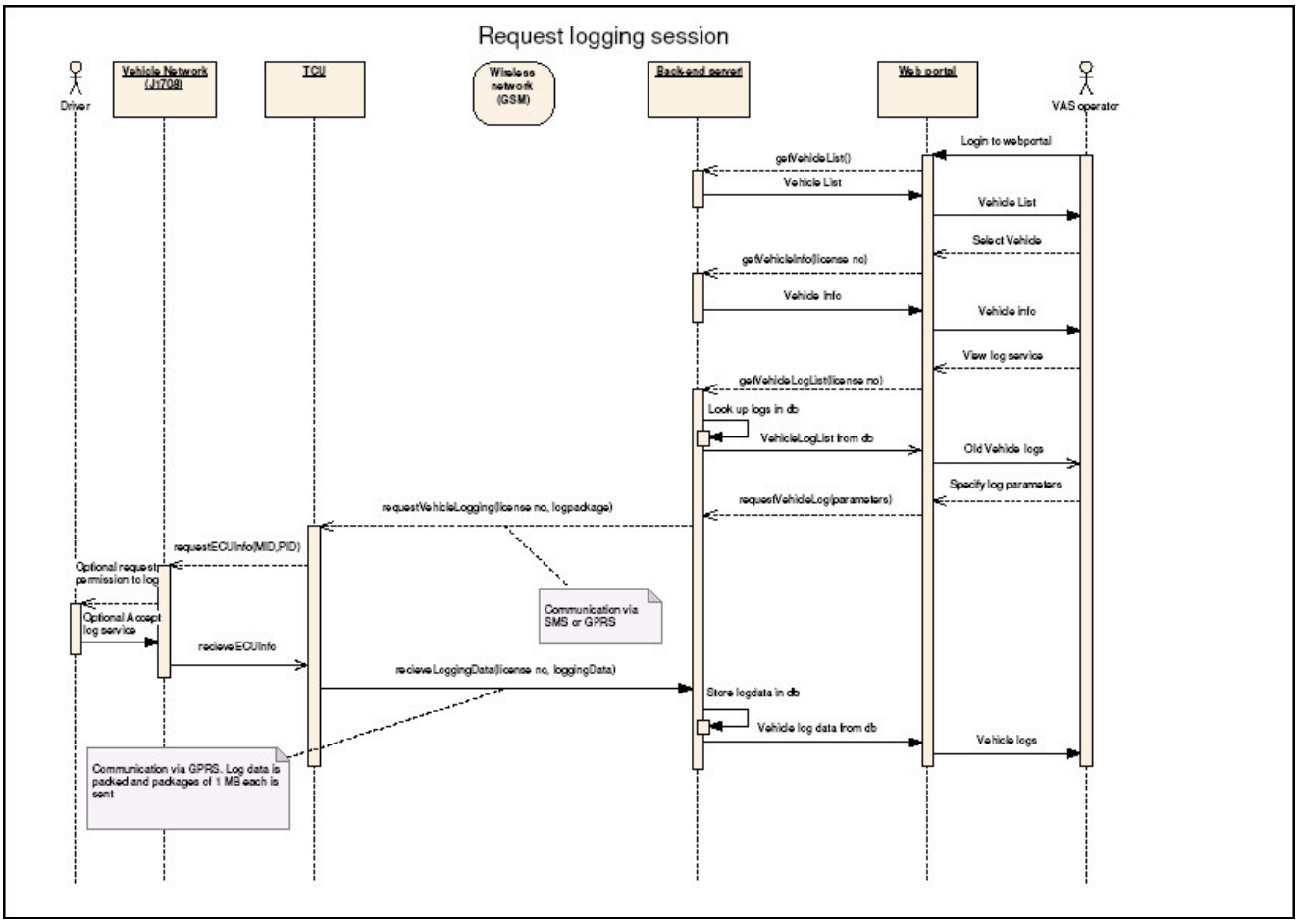


Figure 3: Sequential diagram of request of logs from vehicle

vehicle receives the request the TCU starts to log the parameters on the diagnostic bus in the vehicle and stores the log. When the logging is done the TCU sends the log data to the back-end. The back-end stores the log data in the database. The operator can then update the list in the service application and get the log data presented in a diagram.

4.1.4 Users of the system

This system has one primary user but can be expanded to include more users that could benefit from it. Those are specified as optional and possible future users.

4.1.4.1 Primary user

This service is above all intended for users at the repair shop's call center or similar. The user interacts with the web portal which could be done at any internet connected computer.

4.1.4.2 Optional and possible future users

- Vehicle driver

  The vehicle driver could interact with the system in two possible ways:

  1. Give (or deny) permission for the call center operator to perform remote diagnostic service of the vehicle.
  2. Interact with the vehicle through some kind of Graphical User Interface (GUI) to read error codes and collect parameter information.

- Fleet owner

  Fleet owner may want to get status from their vehicle. They could interact with the vehicle in the same way as the call centre operator – through a web portal similar to (or integrated with) Dynafleet Online.

5 External interface requirements

5.1 TCU user interface

As little interaction with the driver as possible, based on recommendation from Volvo Truck Center personnel. Therefore, a graphical user interface is not needed.

5.2 Hardware interface

5.2.1 Handheld computer

The service can be run on a handheld computer such as an iPAQ or a built-in telematics control unit.

5.2.2 Connection to vehicle network

The handheld computer will need an interface to the vehicle diagnostics network (ie J1708/J1587).

5.3 Communication interface

5.3.1 No contact with server.

The user should be notified if there is no contact with the server.

6 System feature requirements

6.1 Back-office

The back-office should provide information about vehicles in a way that is easy to overview and easy to interact with. The graphical user interface should be easy to understand and the time-for-learning should be short. It must also be possible to find specific information about the vehicle in a convenient way. The user should be able to use the system with basic knowledge about computers and how to use internet.

6.1.1 Vehicle selection

User should be able to select a vehicle by clicking on the license number in a list. A textbox is used in order to filter the list. The list is filtered when the user types a string in the box. When a vehicle is selected, the service menu screen is updated. The vehicle list can be updated by pressing an “Update Vehicle List” button. This is used to update the list after the application is loaded.

6.1.2 Information view

Shows static information about the vehicle from the back-end database such as:

- Description
- VIN
- Make and model
- Owner information

Optional - Shows dynamic information requested from the vehicle such as:

- Current mileage
- Current geographical position

Optional – Service history for the vehicle. The job cards from VCADS could be displayed here.

6.1.3 Notes view

In the notes view, the user can enter notes about the vehicle or other relevant issues. A save button is presented where the user can save the notes.

Optional – Ask the user if the note should be saved if the user selects a new vehicle without saving changes first.

6.1.4 Error codes view

In the error codes view the user can request the error codes for the selected vehicle. It is also possible to request an update of the current view. If available, the current error codes are shown with time stamps.

Optional – The error codes are interpreted to a human readable format.

Optional – Indication of outstanding requests is shown.

6.1.5 View log view

A list of available executed logs is shown. It is possible to display log data, either in numerical or graphical format.

Optional - When new logs are available the list is updated (upon request from the back-end).

Optional – Median, average, min and max value are shown for each parameter.

Optional – Possibility to export data to an external format.

6.1.6 Request log view

A list of predefined log packages is shown. Interval and start time can be set.

It is possible to send a log request with the selected parameters, requesting the log packages to be run at the TCU.

Optional – View scheduled logs in a list. Possibility to remove (cancel) scheduled logs.

6.2 Back-end

Specifications are explained using function calls. It is not a requirement that the back-end is implemented using function calls.

6.2.1 Vehicle database

The vehicle database shall contain the following fields:

- License number as a variable length string
- Phone number to the vehicle TCU as variable length string
- *Optional* - Vehicle Identification Number (VIN)
- *Optional* - IP-number/DNS-name

6.2.2 Listing of vehicles

Ability to provide a list of all the license numbers of the vehicles in the database.

6.2.3 Vehicle information

Ability to provide vehicle information such as VIN and phone number for a given vehicle.

6.2.4 List stored DTCs for a vehicle

List the stored DTCs for a given vehicle.

6.2.5 Request DTCs

Request that active DTCs for the given vehicle be retrieved from the TCU and stored in the database

6.2.6 List stored logs

List the stored logs for a given vehicle.

6.2.7 Get log data

Return data and parameters of a given log.

6.2.8 Get notes

Ability to return notes for a given vehicle.

6.2.9 Store notes

Store given notes for a given vehicle.

6.2.10 Run log package

Store the log package and send the logging request to given vehicle.

6.2.11 Store logging data

Ability to recieve logging result data from the vehicle and store it in the database.
Optional - Notify the back-office that an update is available.

6.2.12 Store DTCs

Ability to recieve DTCs from the vehicle and store it in the database.

Optional - Notify the back-office that an update is available.

6.2.13 Optional - Deploy logging service/bundle

Ability to automatically deploy the diagnostics service to given vehicle.

6.2.14 Optional - Set/View compression settings

Ability to configure and view compression settings for transfers.

6.3 TCU

6.3.1 Upload error codes

On request from back-end the TCU shall upload error codes

6.3.2 Run logging service

On request from back-end the TCU shall perform a logging service. It shall log the requested parameters and store them in a buffer.

6.3.3 Upload logging data

When the logging is done the TCU shall upload the data in the buffer to the back-end.

6.3.4 Optional - Receive log package request

Optional – The TCU shall be able to log a number of different parameters at the same time.

6.3.5 Optional – Cancel logging request

Optional – On request from back-end cancel a logging service.

10. Appendix B – Service analysis table

Huvudkategori	Tjänst	Low BW	Med BW	Hi BW	Time critical	Peri-odic	n/ month	tr/ occ	vol/ tr	Up or down	GPS	CAN	Cam	Local Storage	Screen	LCD	Simple input	Key-board	Audio in	Audio out	Voice channel	RFID	Route planning
Rescue	Rescue + lo	1	0	0	1	0		2	kB	up	1	op	0	0	0	0	1	0	0	0	op	0	0
Rescue	Rescue + me	0	1	0	1	0		-		up	1	op	op	0	0	0	1	0	0	0	op	0	0
Rescue	Rescue + hi	0	0	1	1	0		-		up	1	op	1	0	0	0	1	0	0	0	op	0	0
Diagnostics	Diag Batch	1	0	0	0	0		4		up	0	1	0	op	0	0	0	0	0	0	op	0	0
Diagnostics	Diag Realtime	0	0	1	1	0		4		up	0	1	0	0	0	0	0	0	0	0	op	0	0
Infotainment	Mediaplayer	0	0	1	0	0		2		down	0	0	0	op	1	0	1	0	0	1	0	0	0
Infotainment	Friendsfinder	1	0	0	0	1		2		up/down	1	0	0	0	op	1	1	0	0	0	0	0	0
Infotainment	Messaging	1	0	0	0	0		2		up/down	0	0	0	0	op	1	1	op	0	0	0	0	0
Infotainment	IP Phone	0	1	0	1	0		>2		up/down	0	0	0	0	0	0	1	0	1	1	0	0	0
Infotainment	Video Phone	0	0	1	1	0		>2		up/down	0	0	1	0	1	0	1	0	1	1	0	0	0
On Call	SVT	1	0	0	1	op		2		up	1	0	0	0	0	0	0	0	0	0	0	0	0
On Call	RDU	1	0	0	1	0		1		down	0	1	0	0	0	0	0	0	0	0	0	0	0
On Call	RA	1	0	0	1	0		2		up	op	1	0	0	0	0	1	0	0	0	1	0	0
On Call	TN	1	0	0	1	0		2		up	1	1	0	0	0	0	0	0	0	0	0	0	0
Coverage	Coverage logging	1	0	0	0	1		1		up	1	0	0	op	0	0	0	0	0	0	0	0	0
Coverage	Coverage prediction	1	0	0	0	0		2		down	1	0	0	op	0	0	0	0	0	0	0	0	1
Goods identification/tracking	View goods on truck	1	0	0	0	0		2		up	0	0	0	0	0	0	0	0	0	0	0	1	0
Goods identification/tracking	Report goods pickup/delivery	1	0	0	0	0		2		up	0	0	0	0	0	0	0	0	0	0	0	1	0
010:Remote SW/parameter-up	011:Remote SW campaign of v			1	0	0	0,17	4	500000		0	1											
010:Remote SW/parameter-up	012:Remote SW install of vehic			1	0	0	0,08	4	500000		0	1											
010:Remote SW/parameter-up	013:Remote parameter change			0	0	0	0,17	2	1000		0	1											
010:Remote SW/parameter-up	014:Remote parameter change			0	0	0	0,17	2	1000		0	1											
020:Remote diagnostics (from	021:Remote diagnostics Lvl 1:			0	0	0	0,25	2	250		0	1											
020:Remote diagnostics (from	022:Remote diagnostics Lvl 2:			0	0	0	0,0833	2	2200		0	1											
020:Remote diagnostics (from	023:Remote diagnostics Lvl 3:			1	1	0	0,04	2	400000		0	1											
030:Personal safety	031:Assault alarm/Manual eme			0	1	0	0,04	2	100		1	0											
030:Personal safety	032:Automatic emergency requ			0	1	0	0,04	2	70		1	0											
030:Personal safety	033:Breakdown alarm (request			0	0	0	0,04	2	250		1	1											
040:Vehicle security (requires	042:Stolen vehicle/boat trackin			0	0	0	0,04	2	100		1	0											
040:Vehicle security (requires	043:Immobiliser (when outside			0	0	0	0,17	1	50		1	1											
040:Vehicle security (requires	044:Theft notification (message			0	1	0	0,04	1	50		1	0											
050:Vehicle status monitoring	051:Vehicle operational data fo			0	0	1	1	2	100		0	1											
050:Vehicle status monitoring	053:Snapshot of configuration/			1	0	1	1	2	2000		0	1											
060:Usage monitoring (usage	061:Warranty follow up (downlo			1	0	0	0,04	2	2000		0	1											
060:Usage monitoring (usage	062:Leasing terms (download u			0	0	0	0,17	2	2000		1	1											
060:Usage monitoring (usage	063:Cost per X - pay/damage l			0	0	1	1	2	500		(y)	1											
060:Usage monitoring (usage	064:After market sell-in-opportu			1	0	1	1	2	1000		0	1											
060:Usage monitoring (usage	065:Equipment data, utilisation			1	0	1	4	2	10000		0	1											
070:Order managment (part of	071:Order management: Send			0	0	1	100	4	250		1	0											
080:Driver Time managment	081:Driver Time managment (t			0	0	1	10	2	150		0	1											
090:Vehicle position	091:Locate potential customers			0	0	1	1	2	100		0	0											
090:Vehicle position	092:Locate vehicle in own (Vol			0	0	0	0,01	2	100		0	0											
090:Vehicle position	093:Vehicle location based ser			0	0	1	20	2	200		1	0											
100:Passenger/Goods	101:Video streams / images se			1	1	0	4	2	300000		0	0											
100:Passenger/Goods	102:Traffic info to passenger al			0	1	1	3000	1	50		1	0											
110:Driver support (i.e. driver i	113:Dynamic points of interest,			0	(y)	1	40	1	200		1	0											
110:Driver support (i.e. driver i	114:Vehicle related convenienc			0	0	0	0,17	2	200		1	1											
120:Voice channel	121:Vehicle integrated Voice/P			0	0	0	0																
130:Interfaces	131:Access to data from other			0	0	0	1																
130:Interfaces	132:Publish data to other manu			0	0	0	1																

Lower part of the table is compiled from Telematiktjänster.xls written by Johan Sahlström and Greger Landén

11. Appendix C – Detaljplan förlängning

VOLVO

Företagsnamn/**Company name**	Dokumentnamn/**Name of document**		Sida/**Page**
Volvo Technology Corporation	Detaljplan Förlängning		1 (11)
Utfärdare (avd nr, namn, tfn, geografisk placering, sign)/Issuer (dept, name, phone, sign)	**Datum/Date**	Bilaga/**Appendix**	Reg.nr/**Reg. No.**
06600, Mikael Johansson, Patrick Stenberg, Robert Laxing 031-669433, M1:6	2/9/2006		
Ärende/**Subject**			Utgåva nr/**Issue No.**
Detaljplan för förlängning av examensarbete			1.3

Detaljplan Förlängning

Detaljplan för förlängning av examensarbete

Version

1.3

VOLVO

Volvo Technology Corporation

06600, Mikael Johansson, Patrick Stenberg, Robert Laxing 031-669433, M1:6

1 Introduktion

På grund av förseningar och beroenden av tredjepartsleveranser har vårt examensarbete inte gått att fullfölja helt och hållet. För att kunna få göra en fullständig produkt krävs en förlängning av examensarbetet. Vi har tagit fram en detaljplan för vad som skulle behövas göra och vad som kan tänkas läggas till i produkten som extra funktionalitet.

Uppgiften var från början att göra en tjänst som stämde in på kraven som specificerats inom GST projektet för att kunna utvärdera detta projekt, men på grund av förseningar av hårdvara och mjukvara har en del kompromisser blivit nödvändiga. Vi anser att en anpassning till GST har hög prioritet och därför har vi delat in uppgifterna i tre delar:

- GST-anpassning
- Omarbetning av befintlig mjukvara
- Extra funktionalitet

Vi har brutit ned dessa huvudgrupper i mindre delar och gjort en ungefärlig uppskattning av tidsåtgång för de olika delarna. För att göra GST-anpassning krävs inget från de övriga delarna, men för att göra vissa av de extra funktionerna krävs en omarbetning av befintlig mjukvara.

2 Detaljplan

För att examensarbetet skall kunna resultera i en fullständig produkt krävs modifikationer. Hittills har endast en prototyp utvecklats. Vissa modifikationer är nödvändiga för att kunna konkurrera i en eventuell tävling inom GST om vi blir uttagna till final. Extrafunktionerna är kanske inte nödvändiga för att kunna konkurrera i en tävling men skulle ge en mer konkurrenskraftig produkt.

2.1 GST-anpassning

GST-anpassningen är essentiell för att kunna tävla med produkten. Anledningen till att vi inte implementerat produkten helt och hållet inom riktlinjerna för GST är att referensimplementationen inom GST inte varit klar samt att dokumentation och specifikationer kommit sent. Detta på grund av att vår tidsplan inte varit helt i fas med GSTs tidsplan. Vi har tagit fram några uppgifter som måste lösas för bättre överensstämmelse med GST specifikationerna skall uppnås.

2.1.1 Studie av referensimplementation

För att kunna använda oss av referensimplementationen måste denna studeras i detalj. Det måste läsas specifikationer, men framför allt dokumentation av referensimplementationen. Eventuellt behövs möten med Gatespace som är leverantör.

2.1.2 Software Development Kit (SDK)

Krav för att får utvecklingen av GST-anpassade tjänster. Detta har vi frågat ERTICO efter vid ett tidigare tillfälle med inte fått tag på ännu. Enligt deras hemsida skulle även en simulator finnas.

2.1.3 Deployment, provisioning.

Denna uppgift kan gå snabbt så länge det bara gäller att vi packar våra program (bundles) på rätt sätt och lägger upp på en befintlig webportal eller server. Skall däremot vi stå för en webportal från vilken tjänsten skall laddas ner blir det mer jobb och tar längre tid.

2.1.4 Kommunikation via Connection Manager

Enligt GST:s specifikationer bör TCU:n koppla upp sig med hjälp av en Connection Manager som skall levereras av Prosyst. Prototypen som vi implementerat har inte denna funktionalitet utan använder sig av en egen uppkoppling.

2.1.5 Autentisering

Denna del kan vara tidkrävande beroende på hur väl säkerhetsaspekterna är specificerade i GST och hur väl det är beskrivit hur man skall implementera dessa delar och hur många nivåer som skall finnas. Framförallt är denna del svår att testa men testning är troligtvis inte inkluderat i utvecklingen av produkten.

2.1.6 Utnyttja GSTs system för att läsa ut data

För att GST skall fungera med provisioning och deployment så måste information om varje TCU som är ansluten till GST finnas lagrat. Denna information skulle vi kunna använda för att få reda något om fordonet istället för att hålla en egen databas.

2.1.7 Beräknad tidåtgång för GST-anpassning

För dessa delar har vi beräknat minst 20 mandagar (160 mantimmar), men då gäller det att vi har tillgång till all dokumentation, specifikationer, SDK och referensimplementationer utan att behöva leta någon längre tid efter dem och att dessa är lätta att hantera.

2.2 Omarbetning av befintlig mjukvara

För att kunna göra nya funktioner mer effektiva samt att göra mjukvaran mer robust krävs viss omarbetning av mjukvaran. Den mjukvara som utvecklats hittills är enbart en prototyp med bristfälligt felhantering och effektivitet. Vi har tagit fram några förbättringar som skulle vara lämpliga för att omforma prototyp till produkt.

2.2.1 Omarbetning av protokoll

Det textbaserade TCP-protokoll som vi använder oss av nu är inte bra. Det hade varit bättre att använda oss av UDP vilket är effektivare. Sättet protokollet är uppbyggt på gör det svårt att utöka, ett mer flexibelt protokoll behövs.

2.2.1.1 Beräknad tidsåtgång

5-10 mandagar (40 – 80 mantimmar).

2.2.2 Felhantering

För att få ett mer robust och tillförlitligt system krävs mer och noggrannare felhantering. Detta hör mycket ihop med testning av systemet för att komma fram till vilka oväntade fel som kan uppstå.

2.2.2.1 Beräknad tidsåtgång

Felhantering är något som görs under tiden man utvecklar funktioner och system så vi har inte beräknat någon tidsåtgång för denna uppgift.

2.2.3 Kompilerad kod på back-end

För att optimera systemet och göra det mer effektivt skulle koden behöva omarbetas då att vissa program som idag kompileras vid varje körning enbart behöver kompileras en gång. Detta innebär att skapa en war-fil istället för att använda en jws-fil som idag.

2.2.3.1 Beräknad tidsåtgång

2 mandagar (16 mantimmar)

2.2.4 Schemaläggning av loggar

För att få en effektivare resurshantering behöver köhanteringen av loggningstjänsten omformas. Vi skulle vilja ha en tråd som sköter loggning och kollar vilka loggningar som ligger i kön.

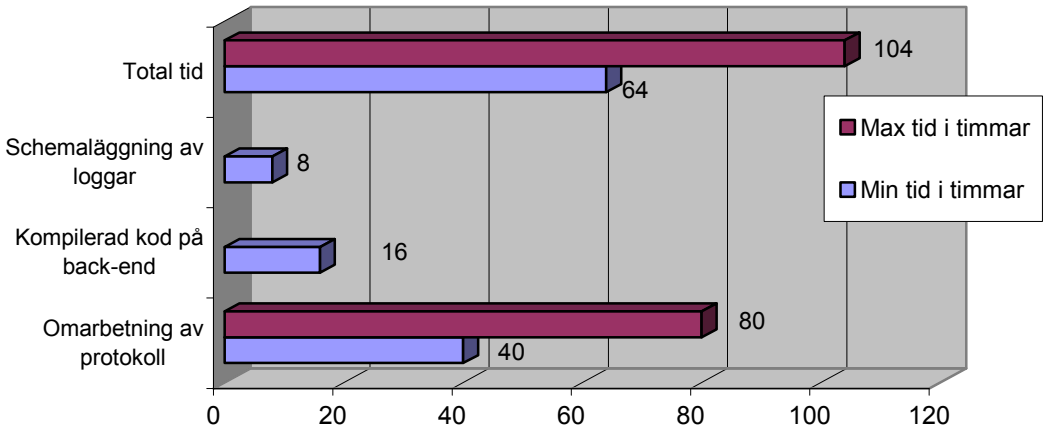
2.2.4.1 Beräknad tidsåtgång

1 mandag (8 mantimmar)

2.2.5 Uppställning av beräknad tidsåtgång

	Min tid i timmar	Max tid i timmar
Omarbetning av protokoll	40	80
Kompilerad kod på back-end	16	
Schemaläggning av loggar	8	
Total tid	64	104

Beräknad tidsåtgång för omarbetning av befintlig mjukvara



2.3 Nya funktioner

En mängd nya funktioner skulle kunna läggas till. Vi har tagit fram en del funktioner som vi skulle kunna utöka systemet med. Vi har försökt att uppskatta tiden det skulle ta att implementera funktionerna för att det skall vara lättare att bestämma vilka funktioner som är rimliga att införa vid en förlängning av examensarbetet. En del av funktionerna är beroende av andra funktioner, ändringar i strukturen av system eller av utomstående resurser, vilket gör det svårare att uppskatta tiden för dessa.

2.3.1 Realtidsloggning

Med en bra uppkoppling är realtidsloggning möjlig. GPRS som vi använder räcker till för att logga i realtid och få upp resultatet av loggarna i mätare och grafer direkt i back-office. Ett exempel på detta vore att logga varvtalet på lastbilen och presentera varvtalet direkt som en varvtalsmätare i back-office.

Här skulle även bandbreddmedvetenhet kunna användas. Då skulle man kunna begära en loggning av flera parametrar som presenteras i olika grafer och mätare i back-office men skulle bandbredden minska så stängs några av mätarna eller graferna av för att spara bandbredd.

2.3.1.1 Beroenden

Realtidloggning är beroende av att ett nytt effektivare protokoll för kommunikationen utvecklas.

2.3.1.2 Beräknad tidsåtgång

15-30 mandagar (120-240 mantimmar)

2.3.2 Outstanding loggs

Om man har behov av att skicka flera loggningsförfrågningar efter varandra som antingen startas samtidigt eller vi olika tidpunkter vore det trevligt att kunna se vilka loggningsförfrågningar som skall köras. Detta skulle kunna lösas med lite ändringar i databasen och i back-office.

2.3.2.1 Beräknad tidsåtgång

2 mandagar (16 mantimmar)

2.3.3 Verktyg för databas

Som back-end är implementerat nu finns inget verktyg för att hantera databasen. Det skulle behövas ett verktyg med funktioner för att rensa gamla loggar, städa upp i databasen och ta bort korrupt data mm. Det skulle även behövas funktioner för att lägga till eller ta bort fordon i databasen samt att uppdatera fordonsegenskaper hos befintliga fordon.

2.3.3.1 Beräknad tidsåtgång

5 mandagar (40 mantimmar)

2.3.4 Triggers – selektiv loggning

Triggers är en funktion som kan vara av både enkel och avancerad karaktär. Vid vissa loggningar kan triggers vara önskvärt. Funktionellt innebär det att om man till exempel vill ha en logg på en temperatur så vill man bara logga när temperaturen avviker från normaltemperatur. Andra parametrar kanske man bara vill logga när de ligger inom ett visst intervall.

En mer avancerad funktion kan till exempel vara att man vill logga kompressionen i cylindrarna, men bara när varvtalet på motorn är mellan 2000 och 3000. Detta skulle också kunna vara en tjänst som hela tiden körs i bakgrunden och skickar ett meddelande till verkstad eller åkeriägare om värdena på fordonet blir dåliga.

Selektiv loggning kan också vara bra då man vill spara bandbredd. Ett exempel kan vara att bara skicka upp data då en parameter ändras, dvs loggar vi en temperatur kanske den inte ändras så ofta utan ligger mestadels konstant.

Denna funktion kan vara mycket användbar vid uppföljning av reparationer. Om en verkstad åtgärdat ett fordon med ett visst problem kan selektiv loggning användas till att kontrollera att problemen med fordonet eliminerats.

2.3.4.1 Beroenden

För att kunna införa denna funktion skulle ett möte med en expert på diagnostik vara nödvändigt. Vi skulle även behöva utveckla ett nytt protokoll för kommunikationen.

2.3.4.2 Beräknad tidåtgång

5 -20 mandagar (40 – 160 mantimmar)

Tidsåtgången av denna funktion beror mycket på hur avancerad man vill att funktionen skall vara samt hur mötet med en eventuell diagnostikexpert avlöper.

2.3.5 Flera typer av parametrar

Prototypen som utvecklats klarar enbart en typ av data. I J1587 specifikationerna finns olika typer av data som kan returneras som i huvudsak skiljer sig åt till antalet bytes i datafältet. I diagnostik-databasen för J1587 finns 10 olika typer och vi har bara implementerat stöd för typ 2. För att kunna implementera en fullständig diagnostik-tjänst krävs att vi stöder alla typer. *Notering: 2006-02-07 implementerades stöd för typ 3-6. Typ 7 verkar inte användas. De andra typerna (1, 8, 9) är bitmönster/flaggor.*

2.3.5.1 Beroenden

För att kunna implementera denna funktion måste databasen omstruktureras och det textbaserade TCP-protokollet omarbetas.

2.3.5.2 Beräknad tidsåtgång

5 mandagar (25 mantimmar)

2.3.6 Loggningspaket

För att kunna diagnostisera ett fordon krävs ibland att flera parametrar loggas samtidigt. På verkstäder har de speciella test som körs på fordonet som loggar flera parametrar under en körning för att få fram status på fordonet För att göra detta möjligt måste loggpaket kunna skapas.

2.3.6.1 Beroenden

Denna funktion är beroende av ett möte med en diagnostik expert och man skulle möjligtvis kunna införa några av de tester som finns i VCADS.

2.3.6.2 Beräkna tidsåtgång

1-4 mandagar (8 – 32 mantimmar)

2.3.7 Namngivning av loggar

En enkel men mycket användbar funktion är att kunna namnge loggarna. I prototypen namnges loggarna med tidstämpel, men det gör det svårt att hitta en specifik logg om man inte vet exakt när den kördes.

2.3.7.1 Beräknad tidsåtgång

1 mandag (8 mantimmar)

2.3.8 Fjärrstyrning

Fjärrstyrning skulle kunna vara en användbar funktion om man under en diagnostisk session skulle behöva ändra varvtalet eller liknande. För att kunna införa denna funktion måste en högre säkerhet användas och en studie av vad som får skrivas på bussen måste genomföras. Det finns andra avdelningar på Volvo Technology Corporation som har implementerat fjärrstyrning så man kan studera deras arbete.

Ur demosynpunkt kunde detta vara en mycket användbar funktion med hög underhållningsfaktor.

2.3.8.1 Beroenden

Denna funktion är mycket beroende på vad som gjorts tidigare och hur mycket rättigheter vi har för att skriva på fordonsnätverket. Utan tillräckliga rättigheter kan inte denna funktion implementeras.

2.3.8.2 Beräknad tidsåtgång

Vi har inte beräknat detta på grund av alla beroenden.

2.3.9 Ajax

Med syfte att få ett snyggt användargränssnitt direkt i en webläsare skulle Ajax kunna användas. Idag har vi problem med att vår Back-office applikation inte får tillräckliga säkerhetsrättigheter i webläsarna, vilket innebär att vi inte kan köra den som en applet. Med Ajax hoppas vi kunna komma runt de problemen och back-office applikationen skulle kunna bli mer flexibel och lättillgänglig. Vi har dock inte jobbat med Ajax tidigare vilket innebär vissa förberedande studier innan back-office applikationen kan portas till Ajax.

2.3.9.1 Beräknad tidåtgång

25 mandagar (200 mantimmar)

2.3.10 Integration med hårdvaruplattformen för GST-testsiten i Göteborg

Enlig den ursprungliga planen för examensarbete skulle vi använda oss av den hårdvara som skall användas på GST-testsiten i Göteborg. Denna blev dock försenad så vi fick överge de planerna och implementera systemet för Pocket PC på en IPAQ istället. GST-hårdvaran är dock en mer ändamålsenlig plattform och det vore därför önskvärt att få möjligheten att porta vår applikation.

2.3.10.1 CAN-stöd

Det hade även varit önskvärt att införa CAN-stöd. Detta blir dock lite omständligt att göra på en Pocket PC eftersom ny hårdvara måste tillföras. Detta har vi därför lagt som en del av integrationen med GST-hårdvaran eftersom vi då kommer ha tillgång till CAN genom hårdvaran.

2.3.10.2 Beroenden

Integrationen med GST-hårdvaran är naturligtvis beroende av hur det går i det projektet, framförallt vad gäller hårdvara och javamotor.

2.3.10.3 Beräknad tidsåtgång

30 – 50 mandagar (240 – 400 mantimmar)

2.3.11 Komprimering av loggar

För att spara bandbredd kan komprimering av loggar införas. Detta kan göras på flera olika sätt. Ett sätt kan vara att omforma meddelandeformatet så att meddelandena som skickas över GPRS hålls korta. Man kan även komprimera meddelande genom att använda en befintlig komprimeringsalgoritm som till exempel zip.

2.3.11.1 Beroenden

Även denna funktion är beroende av en omformning av protokollet för överföring av meddelanden.

2.3.11.2 Beräknad tidsåtgång

10 mandagar (80 mantimmar)

2.3.12 Abstrahering av hårdvaruberoenden

Man skulle kunna dela upp tjänsten i flera bundles, där all hårdvaruberoende kos samlas i en enda bundle med ett väldefinierat interface mot resten av bundlarna. Detta gör det enkelt att anpassa mjukvaran för andra hårdvaror. För att göra detta behöver diagnostikförfaranden generaliseras och då behövs utökad förståelse för diagnostik och kunskap om fler protokoll än J1587.

2.3.13 Stöd för SMS

SMS-stöd är en viktig funktion eftersom det kanske inte alltid finns gprs-täckning. Det går alldeles utmärkt att extrahera DTCer via SMS och även i viss mån överföra loggdata. För att erbjuda stöd för SMS behövs dels stöd i bilen (vilket kan lösas relativt enkelt genom koppling till GST-hårdvaran), dels stöd på serversidan, antingen via en sms-gateway på annan server eller hårdvara kopplad direkt till vår server.

2.3.13.1 Beroenden

Bör lösas genom integration med GST-hårdvaran.

2.3.14 Uppställning av beräknad tidsåtgång

Tabellerna innehåller tidsåtgång i timmar

Funktion	Min tid	Max tid
Realtidsloggning	120	240
Outstanding logs	16	
Vertyg för databas	40	
Triggers	40	160
Flera typer	40	
Loggningspaket	8	32
Namngivning av logg	8	
Abstrahering av hårdvaruberoenden		
Stöd för SMS		
Fjärrstyrning		
Ajax	200	
Integration med DF EVO inkl. CAN-stöd	240	400

3 Resurser

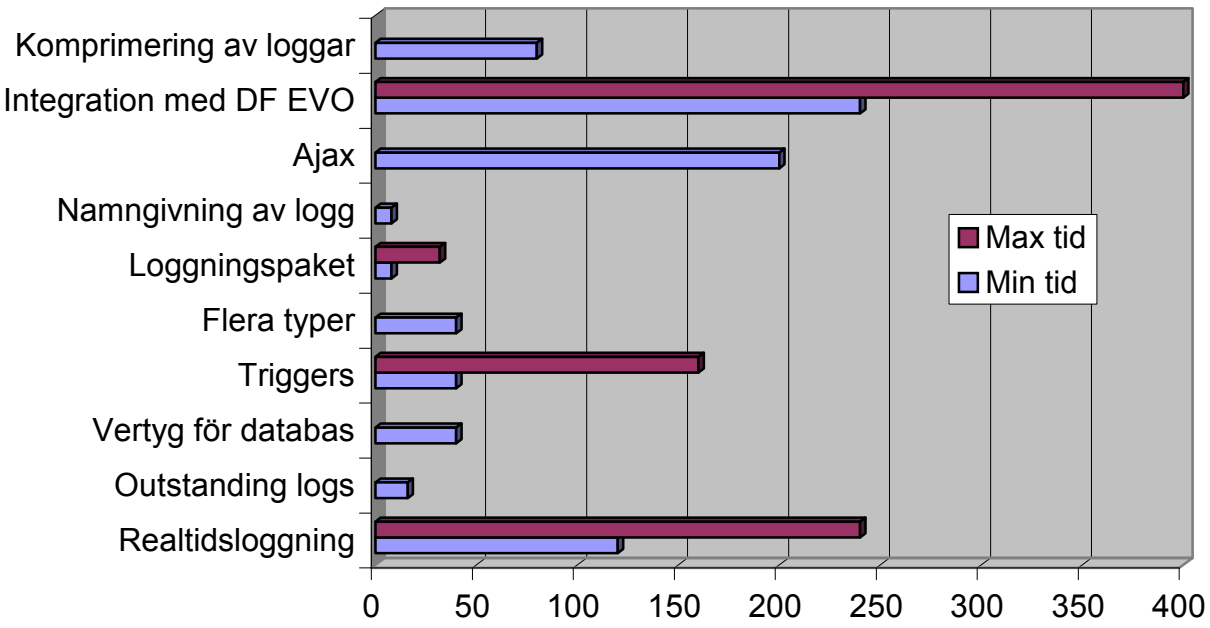
För att arbetet skall resultera i ett lyckat resultat behövs vissa resurser utifrån användas. En del av dessa har redan nämnts ovan eftersom det krävs viss experthjälp för att kunna införa vissa funktioner. Några resurser som kan behövas användas beskrivs här.

3.1 Expertis

Beroende på vilka funktioner som skall införas och hur produkten skall presenteras osv kan vissa personer behövas. Skall GST-hårdvaran användas kan en del hjälp av Marcus Larsson behövas. Behöver vi tillgång till LSP för presentationer kan David Rylander vara till bra hjälp. För att införa mer avancerade diagnostiska funktioner kommer vi även att behöva konsultera experter inom diagnostik.

3.2 Testare

Produkten kommer att behöva testas och verifieras. Detta kan skötas antingen av oss själva med även med hjälp av utomstående testare. Utomstående testare är att föredra



men då måste resurser för detta också bokas upp.

3.3 Hjälpmedel

Under examensarbetet har datorkraften varit på gränsen att inte räcka till, vi har varit tvungna att installera mer minne i en av datorerna och den bärbara dator som finns

tillgänglig är för trött för att den skall vara effektiv att jobba med. Mer kraftfulla datorer vore att föredra. Dessutom vore bättre skärmar att föredra.

3.4 Stöd för andra examensarbeten

På grund av vår kunskap inom OSGi och GST har vi redan fått förfrågan att vara behjälpliga för andra examensarbetare. Detta är inget som är klart ännu men skulle så vara fallet så får lite extra tid medräknas för handledning av kommande examensarbeten.

4 Tidsummering

Att summera de beräknade tiderna för alla specificerade uppgifter ger ingen exakt siffra med kan ge en uppfattning om vilken tidsplan vi kan få. Vi har valt att summera alla uppgifters minimumtider och deras maximumtider för att få fram ett tidsintervall. Maximumtiderna är medvetet väl tilltagna och alla uppgifter kanske inte skall genomföras.

	Minsta tid alla uppgifter	Max tid alla uppgifter
GST anpassning	20	200
Omarbetning mjukvara	64	104
Nya funktioner	792	1216
Total tid i timmar	**876**	**1520**
Total tid i veckor	21,9	38