Architecture for a remote diagnosis system used in heavy-duty vehicles

by

Anders Björkman

LIU-IDA/LITH-EX-A--08/034--SE

2008-06-30
Final Thesis

Architecture for a remote diagnosis system used in heavy-duty vehicles

by

Anders Björkman

LIU-IDA/LITH-EX-A--08/034--SE

2008-06-30

Supervisor: Jan Lindman
Scania Fleet Management Department at Scania CV AB
Examiner: Petru Eles
Dept. of Computer and Information Science at Linköpings universitet
Abstract

The diagnosis system of a Scania vehicle is an indispensable tool for workshop personnel and engineers in their work. Today Scania has a system for fetching diagnostic information from field test vehicles remotely and store them in a database, so called remote diagnosis. This saves the engineers much time by not having to visit every vehicle. The system uses a Windows based on-board PC in the vehicle called an Interactor. The Interactor has a telematic unit for communication with Scania’s Fleet Management System and the CAN-bus in the vehicle. In the next generation of the Interactor, its telematic unit is to be replaced by a Linux based telematic unit called the Communicator 200 (C200). The purpose of this master project is to create a new architecture for a remote diagnosis system that uses the new telematic unit Communicator 200.

The thesis gives an analysis of the current remote diagnosis system used at Scania and proposes an architecture for a new generation remote diagnosis system using the C200. Also a system for demonstrating how to perform remote diagnosis over the C200 has been built. The thesis describes the operation and how the demonstration system was implemented.

Keywords: Remote diagnosis, CAN, GPRS, DTC, Fleet Management
Contents

Introduction ........................................................................................................................................... 1
  1.1 Background .................................................................................................................................. 1
  1.2 Project definition ......................................................................................................................... 2
  1.3 Resources ................................................................................................................................... 2
  1.4 Method ....................................................................................................................................... 2
  1.5 Limitation ................................................................................................................................... 3
  1.6 Outline ...................................................................................................................................... 3

2 Underlying technologies .................................................................................................................. 5
  2.1 The Internet Protocol suite ....................................................................................................... 5
  2.2 Controller area network, CAN .................................................................................................... 6
  2.3 Wireless communication ........................................................................................................... 7
  2.4 Linux ......................................................................................................................................... 7
  2.5 Microsoft Message queuing ....................................................................................................... 8
  2.6 Programming languages ........................................................................................................... 8
  2.7 .NET XML Web services ........................................................................................................... 9
  2.8 The Interactor ............................................................................................................................. 10
  2.9 Communicator 200 ..................................................................................................................... 11

3 Existing remote diagnosis system architecture ............................................................................. 13
  3.1 Overview ................................................................................................................................... 13
  3.2 The CAN-bus in a Scania vehicle ............................................................................................... 14
  3.3 Diagnostic protocols .................................................................................................................... 14
  3.4 The Interactor application .......................................................................................................... 16
  3.5 SCOMM .................................................................................................................................... 17
  3.6 Comprehensive functionality ...................................................................................................... 19
  3.7 Other systems for remote diagnosis .......................................................................................... 20

4 An architecture for a remote diagnosis system ............................................................................. 21
  4.1 Goals with a future system ........................................................................................................ 21
  4.2 Overview of the system .............................................................................................................. 21
  4.3 Communication between the FMP and the vehicle ................................................................... 22
  4.4 Suitable scripts .......................................................................................................................... 22
  4.5 How to utilize SCOMM .............................................................................................................. 27
  4.6 Conceivable architecture for the server side ............................................................................. 28
  4.7 Requirements on C200 application ........................................................................................... 30

5 Implementation of a demonstration system .................................................................................... 33
  5.1 The server side ........................................................................................................................... 33
  5.2 The C200 application ................................................................................................................ 38
  5.3 Overview of the complete system ............................................................................................. 40
  5.4 Limitations .................................................................................................................................. 40

6 Conclusions and recommendations .............................................................................................. 43

7 Future work ..................................................................................................................................... 45

References .............................................................................................................................................. 46
Figures

Figure 2.1: The OSI model. ......................................................................................................................... 5
Figure 2.2: Underlying protocols. .................................................................................................................. 9
Figure 2.3: Lifetime of an XML Web service. ............................................................................................... 10
Figure 2.4: A Scania Interactor 300 mounted on the dashboard. ............................................................... 10
Figure 2.5: The C200 “Blackbox” telematic unit. ......................................................................................... 11
Figure 2.6: Block scheme. ........................................................................................................................... 10
Figure 2.7: Layers. ...................................................................................................................................... 11
Figure 2.8: Application structure for C200. ................................................................................................. 12
Figure 3.1: High level overview. ................................................................................................................ 13
Figure 3.2: The CAN-bus connecting all ECUs in a Scania vehicle. ........................................................... 14
Figure 3.3: Generic message flow example ............................................................................................... 15
Figure 3.4: Application components. ......................................................................................................... 16
Figure 3.5: Settings file transfer.................................................................................................................. 17
Figure 3.6: The SCOMM architecture. ...................................................................................................... 18
Figure 3.7: Interfaces to the application. ................................................................................................... 19
Figure 3.8: Architecture of today’s remote diagnosis system. ................................................................. 20
Figure 3.9: A file for a DTC readout generated by CDev. .......................................................................... 20
Figure 4.1: Components on the server side and the vehicle side. .............................................................. 22
Figure 4.2: A sample XML script. ............................................................................................................. 23
Figure 4.3: Example INI file. ..................................................................................................................... 24
Figure 4.4: Example of a JSON representation of a person. .................................................................... 25
Figure 4.5: Example of a YAML script representation of a person. .......................................................... 26
Figure 4.6: Server side architecture ........................................................................................................... 29
Figure 4.7: C200 applications architecture. .............................................................................................. 30
Figure 4.8: Setting up ECUs. ..................................................................................................................... 31
Figure 5.1: The remote diagnosis interface. ............................................................................................... 33
Figure 5.2: Structure for the request message byte vector. ..................................................................... 34
Figure 5.3: The user interface. .................................................................................................................. 35
Figure 5.4: A SCOMM communication log. ............................................................................................. 36
Figure 5.5: A SCOMM demo file. ............................................................................................................... 37
Figure 5.6: The Web service. ..................................................................................................................... 37
Figure 5.7: The communication server. .................................................................................................... 38
Figure 5.8: Request – response scheme. .................................................................................................... 38
Figure 5.9: A KeyWord response message. ............................................................................................... 39
Figure 5.10: The complete demo system. ................................................................................................. 40
Introduction
This final thesis for a Master of Science degree has been performed at the Scania Fleet Management department in Södertälje and corresponds to 20 weeks of full-time work. The work was performed on commission of the development department REIV during the winter of 2007/2008.

1.1 Background

Scania is a large manufacturer of heavy-duty trucks, buses, industrial and marine engines. Like any other truck manufacturer a Scania vehicle has a system for on-board diagnosis (OBD). OBD evolved because of the increasingly complexity of a modern vehicle and to meet the stricter emission standards. An on-board diagnosis system is mandatory for new heavy-duty vehicles in Europe from the first of October 2006 [24].

Diagnosis of a Scania vehicle is typically performed by connecting a laptop to the on-board diagnosis connector in the vehicle. It is then possible to execute diagnosis commands and extract the vehicles Diagnostic Trouble Codes (DTC) from a set of Electrical Control Units (ECU). These ECUs monitor and control different parts of the vehicle, e.g. brakes, engine or gearbox. DTCs are generated in the ECUs when an error is detected. The DTCs can then be extracted using a diagnostic tool for regular maintenance or for fast repair and replacement of faulty components.

Scania Fleet Management is a set of computer based systems, offering services to hauliers, who aim to make vehicle fleet management more effective. Scania Fleet Management is connecting offices and vehicles over the Internet and the Global System for Mobile communication (GSM) network, and thereby making hauliers able to improve communication with, and supervision of, their fleets. Inside the office, this is achieved by either integrating the services with the existing office system or accessing them via a Web portal (The Fleet Management Portal, FMP), while on the vehicle-side this is achieved by installing an on-board PC, called an Interactor.

By using the infrastructure of the Fleet Management system it is possible to send diagnosis commands, as a script, over the communication link and to a vehicle. The on-board computer can upload the response from the diagnosis command and store it in a land-based database. The concept described above is referred to as remote diagnosis.

In the next generation of the Interactor, a new on-board telematic unit, Communicator 200 (C200) is introduced. The C200 replaces the current telematic unit in the Interactor. The intension is to use the C200 for the remote diagnosis part instead. The purpose of this master project is to find an architecture for remote diagnosis based on the telematic unit C200. Part of the project is also to investigate how existing diagnosis software can be reused.

The system described in this thesis is not a new system but merely the third generation remote diagnosis system at Scania. Back in 1997, Scania used the GSM network for fetching diagnostic information from their field test vehicles.
1.2 Project definition

The assignment of the master project can be divided into six parts:
1. Document system architecture and functionality of Scania’s existing system for remote diagnosis.
2. Propose how existing diagnosis software can be reused.
3. Create a complete, script based system architecture for a remote diagnosis system based on the C200 telematic unit.
4. Find a suitable script.
5. Investigate security issues in the new system.
6. Build a demonstration system where arbitrary diagnosis commands can be executed from a user interface. The demonstration system should also be able to fetch DTCs from the vehicle.

A future use case for the remote diagnosis system is that the workshop personnel, before the day of vehicle maintenance, can fetch the DTCs remotely. In this way the mechanics can prepare for the workshop visit by ordering the right spare parts before the vehicle arrives.

1.3 Resources

Since this project is very Scania specific the literature studied has mostly been internal Scania specifications rather than scientific articles. The material in Chapter 2 has been taken from textbooks and internet sources for the corresponding subjects.

The best method for determining systems functionality has been to examine the source code of different Scania software. This since many software components lack documentation and are still under development.

Most focus in the master project has been on implementing the demonstration system.

1.4 Method

First a literature study has been done to identify relevant literature and to study the area of diagnosis. Meetings with engineers at Scania were carried out to get a vision of what the system could look like. The meetings also gave hints on how to get started.

The actual design of the demonstration system can be described as a waterfall model approach. The master project was of an investigating nature and not a software development process. Still the waterfall was an applicable method.

A strategy has been to first try to implement the demonstration system and hope that the architecture described in chapter 4 would emerge from the practical implementation. The actual architecture was not implemented why step five to seven was not applicable in this case.

The standard waterfall model for systems development is an approach that goes through the following steps [25]:
1. Document system concepts.
2. Identify system requirements.
3. Break the system into pieces.
4. Design each piece.
5. Code the system components and test them individually.
6. Integrate the Pieces and Test the System.
7. Deploy the System and Operate It.

Step 1, documenting system concepts have been done in chapters 2 and 3. The system requirements have been documented under section 4.1. They have been identified by interviewing engineers at Scania. Since there was no formal requirements setup for the project, requirements have been called goals.

The demonstration system (see chapter 5) consists of several components (pieces) and was designed and coded individually. Testing has been done to each component and to the system as a whole. The operation and implementation of the demonstration system is documented in chapter 5.

1.5 Limitation
The project definition was very wide and therefore the project has been delimited to be able to keep it within twenty weeks.

- System security has not been thoroughly investigated.
- The demonstration system has some limitations described in section 5.4.
- The thesis suggests changes to the diagnostic software, SCOMM, but does not indicate on how to implement them.

1.6 Outline

Chapter 1 Introduction to the thesis
Chapter 2 Theory of technologies used in the project
Chapter 3 Theory behind the current remote diagnosis system
Chapter 4 Proposed architecture for a new remote diagnosis system
Chapter 5 Implementation and operation of demonstration system
Chapter 6 Recommendations and conclusions drawn from the project
Chapter 7 Future work
2 Underlying technologies

This chapter will introduce the reader to concepts and techniques used in this thesis. The purpose is to provide a frame of reference to understand the topics covered later in this thesis. Most of the techniques are only described at a higher level since the low level implementation is not of relevance to the project.

2.1 The Internet Protocol suite

A network can be described as number of nodes, which are interconnected to each other. The purpose of the network is to be a resource to move data between nodes. To establish connections between the nodes, different types of switching techniques are used, namely circuit switching and packet switching.

In circuit switching a dedicated communication path is obtained between the source and the destination. The path remains uninterrupted for the entire duration of the transmission.

In packet switching messages are subdivided into segments not exceeding a specified size, often called packets. The packets are transmitted over the network, whose resources are shared dynamically by a number of active nodes. Thus the available bandwidth is shared among the active nodes. At the destination node, the original message is reassembled on a packet-by-packet basis.

In order to reduce complexity, the network structure is organized as a set of layers. Every layer has a specific function to perform and it offers services to its neighboring layers. The rules used for communication between a sending layer and a receiving layer are called layer protocols. A model to characterize the network architecture is the OSI (Open System Interconnection) reference model. The main objective with the OSI model is to specify mechanisms for communication between systems in a communication network. In its original form it consists of seven layers as seen in Figure 2.1.

The physical layer is responsible for the transmission of the raw bit stream over the physical medium. The main objective of the data link layer is to ensure reliable transmission of data across the physical link.

Figure 2.1: The OSI model.

The data link layer receives data from the higher level, splits the data into packets and transmits them. In broadcast networks a sub layer in the data link layer called medium access control (MAC) controls access to the shared medium. The network layer is responsible for routing data packets from source node to destination node. A popular protocol in the network layer is the Internet protocol (IP), which is used for communicating data across a packet-switched network. The IP is connection-less meaning that no circuit setup is needed before a host tries to send packets to a host it has previously not communicated with. The IP provides a best-effort service which means that no guarantees are made concerning reliable transfer of a packet. When the transport layer has data to send, it passes the data to its local IP together with the IP address of the intended recipient.

The transport layer could be seen as an interface between the higher and lower layers. The basic function of the transport layer is to accept packets from layers above, split it into smaller blocks,
pass these to the network layer and ensure that the blocks will arrive correctly at the other end. The transport layer also determines what type of service to provide to the session layer. Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are popular protocols used on the Internet that operate in the transport layer. The UDP protocol provides a best-effort service, hence messages or datagrams may arrive out of order, appear duplicated, or go missing without notice. In return UDP does not need the overhead of checking whether every packet actually arrived. This makes UDP faster and more efficient, at least for applications that does not need guaranteed delivery.

The session layer allows users to establish sessions between them. Services carried out at the session layer are for example dialog control, i.e. keeping track of whose turn it is to transmit. The presentation layer is concerned with the syntax and semantics of the information transmitted. Finally, the application layer contains a variety of protocols that are commonly used by the users. This type of interface is generally called Application Programming Interface, API. Protocols that belong to the application layer are e.g. HTTP, SSH and FTP.

2.2 Controller area network, CAN

This section is based on [3]

The CAN-bus was developed in 1988 by Intel Corporation and Robert Bosch. The CAN specification defines the physical level and the data link layer in the OSI model. The layers above the data link layer require additional software protocols, such as DeviceNet or SAE J1939.

Controller Area Network (CAN) is a two-wire, half duplex, high-speed network system that was originally designed for the automotive industry. CAN is especially used for European cars, but has also become a popular bus in industrial automation as well as other applications. The CAN bus is primarily used in embedded systems and, as its name implies, is a network technology that provides fast communication among microcontrollers up to real-time requirements.

While, for instance, TCP/IP is designed for transport of large data amounts. CAN is designed for real-time requirements and is superior to a high speed TCP/IP connection when it comes to short reaction times, timely error detection, quick error recovery and error repair.

In the CAN standard all messages are referred to as frames. The CAN specification provides four different types of message frames:

- **Data frame** – This kind of frame contains the data sent to one or several nodes.
- **Remote frame** – This frame requests data from another source and is followed by a data frame containing the requested data.
- **Error frame** – Any bus participant can send this frame to report error conditions.
- **Overload frame** – If a node is overloaded it can send a delay request between data and a remote frame transmission.

Every node transmitting a data or remote frame will be the bus master during that transmission. The actual bus access is managed through non-destructive bit-wise arbitration, which in turn provides message collision avoidance in case that multiple nodes attempt to access the bus at the same time. CAN data transmissions are distinguished by a unique message identifier (11/29 bit), which also represents the message priority. The length of the message identifier depends on which CAN specification is used. CAN 2b define a 29 bit message identifier and CAN 2a an 11 bit identifier. A low message ID represents a high priority. High priority messages will gain bus access within shortest time even when the bus load is high due to the number of lower priority messages. Frames

---

1 Society of Automotive Engineers
that lost in the arbitration or are interrupted by error will be retransmitted as soon as the medium is sensed idle.

2.3 Wireless communication

When data needs to be sent from a moving target, or from remote locations, wireless communication is desirable. The Scania Fleet Management system uses GPRS over the GSM network with fallback on SMS for data transfer between vehicle and server. The system is designed to buffer messages when the vehicle is in an area with no GSM signal. The messages can then be transmitted when the vehicle enters an area with GSM signal.

2.3.1 GSM

In 1982 the first step in creating an European mobile communication standard was taken by the Confederation of European Posts and Telecommunications (CEPT) and the Groupe Spéciale Mobile (GSM) was born. Their objective was to design a European standard for mobile communication to be used all over Europe. In 1989 the responsibility was shifted over to the European Telecommunications Standards Institute (ETSI) and Global System for Mobile communications (GSM) was accepted as an international mobile digital telephony standard. The first GSM network was launched in 1991. [4]

GSM is based on circuit switching. To allow many simultaneous calls over the radio link, GSM uses multiple frequency channels with 200 KHz separation based on a narrow band TDMA\(^2\) solution and 8 time slots. The GSM system was mainly designed for voice communication, but could also, due to the digital structure, offer data communication with bit rates of 9.6 and 14.4 kbit/s. [1]

2.3.2 GPRS

[This section is based on reference [2] ]

GSM was developed primarily to provide a telephony service for mobile users and the associated data service relied on a connection-oriented packet-switched network. The Internet operates using a connectionless packet mode and access to it through an access gateway or router. To enable enhanced data service an extension to the GSM system called general packet radio service (GPRS) was introduced. GPRS provides and end-to-end connectionless packet service that includes packet-mode transfers over a radio access network and an IP-based packet-switched core network. The components that make up the radio access network are shared dynamically between the traffic relating to both circuit-mode and packet-mode services. In packet-mode services bandwidth is allocated dynamically in both the uplink and downlink, according to demand. The maximum data rate that can be allocated is 171.2 kbit/s, but that data rate is shared among both packet-mode and circuit-mode services.

2.4 Linux

[This section is based on reference [5] ]

Linux is a project that was initialized to create a version of UNIX that could run on Intel-based computers (IBM PC-compatible). Linux was initially a hobby project by Linus Torvalds while he was studying at the University of Helsingfors. The operating system is designed and built from hundreds of programmers worldwide. The goal has been to create a UNIX clone, free from commercial software which anyone can use free of cost.

\(^2\) Time division multiple access. A technique for sharing a common media among several users by assigning each user a time slot to transmit in.
The Linux operating system consists of the programs from the GNU-project, the Linux kernel and other software. The kernel is a monolithic kernel, meaning that most system processes like input, output, memory management and drivers are run as a part of the kernel. This makes for high performance, but if one process fails it may cause the whole system to become unstable.

2.5 Microsoft Message queuing

The Microsoft Message Queuing (MSMQ) technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. Applications send messages to queues and read messages from queues that are stored in the computer memory. [26]

Using message queues has several advantages:

- An application can send messages to the queue at a rate that does not depend on how fast the receiving applications can process the messages.
- Applications sending or receiving messages to and from a queue can fail without affecting each other. If, for example, the receiving application fails, the sending application can continue to send messages to the queue. When the receiver is up again, it can process the messages from the queue.
- Sending applications can overwhelm receiving applications with messages. Queues can manage mismatched message production and consumption rates so that a receiver is not overwhelmed.
- Sending, receiving, and processing operations can become disconnected when communicating over high-latency networks or limited-availability networks, such as in the Scania Fleet Management System. Queues allow these operations to continue, even when the endpoints are disconnected. When the connection is re-established, the queue forwards messages to the receiving application.

There are also problems with using MSMQ. If a message cannot be processed it will block the entire queue. These messages may have to be manually erased using a queue tool. [27]

2.6 Programming languages

Mainly two programming languages have been used when implementing the demo system, namely C++ and C#. C++ is used for the applications in the C200, while C# is used on the server side, where all components were written in .NET. This section will give an introduction to the languages to give the reader an overview of their characteristics.

2.6.1 C++

[This section is based on reference [6]]

C++ is an object-oriented language derived from C. The C++ language was developed at Bell Laboratories by Bjarne Stroustrup in the beginning of the 1980’s. The syntax of C++ is similar to C, with various extensions and extra keywords needed to support classes, inheritance, exception handling, overloading and polymorphism. Because very little need happen behind the scenes in C++ programs, they have relatively low overhead and good performance.

---

3 The GNU Project is a free software, mass collaboration project.
4 Compared to e.g. Java.
Overloading of functions is an important attribute in the C200 application architecture. Overloading is a type of polymorphism where different functions with the same name are invoked based on the data types of the parameters passed. Inheritance is widely used in the C200 applications to enable the reuse of code.

2.6.2 C#

[This section is based on reference [12]]

The first widely distributed implementation of C# was released by Microsoft in July 2000, as part of its .NET Framework initiative.

C# is intended to be a simple, modern, general-purpose, object-oriented programming language. The language has many similarities to C and C++, which makes for easy learning for people already familiar to those languages. The language features e.g. automatic strong type checking, array bounds checking, detection of attempts to use uninitialized variables and automatic garbage collection. Although C# applications are intended to be economical with regard to memory and processing power requirements, the language was not intended to compete directly on performance and size with C or assembly language. However, C# makes for very rapid application development but to the expense of performance.

2.7 .NET XML Web services

[This section is based on reference [7] and [28]]

XML Web services are a category of software components that provide functionality over a network. With an XML Web service, information is available through a well-defined interface and is accessed using standard protocols and data representations. Any device or program, regardless of platform or implementation language, can communicate with the XML Web service across the network to access information.

There are several benefits of using XML Web services. They are standards based which means that implementations operate in the same way, use the same protocols and encode data consistently. This makes it possible to consume a XML Web service regardless of what platform the service and consumer is running on. XML Web services are vendor neutral which means that the core technology is open and neutral for innovations and improvements.

There are situations when XML Web services are not an appropriate method for information exchange. They are not suitable for a system designed for a special group of users or users of a special platform. This because consuming of XML Web services is platform independent. XML Web services should neither be used when system reliability and performance are demanded. They use a large protocol stack with a lot of overhead and the protocols used are the same as those used on the Internet. Also XML Web services is a immature technology which means that it does not provide answers to many issues, such as security, transactions and message routing.

As mentioned, interaction with XML Web services is usually performed by using standard Internet protocols. The protocols used can be seen in Figure 2.2. A description of each protocol follows.

Figure 2.2: Underlying protocols.
• Discovery – Enables the developer to obtain a description of an XML Web service.
• Description – Provides the client with a description of how to interact with the service. The WSDL language is the standard of describing an XML Web service.
• Messaging – Messaging is done by sending SOAP requests.
• Encoding – Encapsulates the XML message into a SOAP message.
• Transportation – To send the message over a network HTTP is used.

The process that occurs when an XML Web service is called is similar to the process that occurs when calling a regular method. The main difference is that instead of calling a method that is located in the client application, the messaging layer generates a request message over the transport layer. Because the XML Web service method can be located on a different computer, the information that the XML Web service needs to process the request must be passed across the network to the server that hosts the XML Web service. The XML Web service processes the information and sends the result back, over the network, to the client application.

In a .NET XML Web service the proxy objects seen in Figure 2.3, can be described as the glue between the .NET framework and an .NET XML Web service. It allows a programmer to easily make use of the service. The proxy is a .NET class that exposes the functionality of the XML Web service via methods. Without a proxy, a programmer must handle all of the interactions with the XML Web service manually.

Figure 2.3 shows the process of communication between a client and a .NET XML Web service.

2.8 The Interactor

[This section is based on reference [18]]

The Interactor or SVIP is a complete on-board hardware platform including a PC, GPS and a telephone module. The Interactor is running Windows XPe (XP embedded) and is mounted on the dashboard of the vehicle. Along with the computer there is a flat touch screen for interaction with the driver. The Interactor has a so called telematic handler board which handles all communication to, from and with the vehicle. The telematic handler board is always powered to enable communication when the vehicle ignition is switched off. With
the Interactor the driver can e.g. send and receive drive orders, call for assistance and send messages to the trucking company.

### 2.9 Communicator 200

This section is based on reference [19]

The Scania Communicator 200 (C200, seen in Figure 2.5) is a vehicle telematic unit, which in the future will replace the telematic unit in the Interactor. It was designed to cope with an increasing need for customer services through the Fleet Management Portal and to support future functionality. It runs the Linux operating system with a Board Support Package (BSP), which contains the drivers for the hardware.

The C200 contains e.g. a GPRS modem, a GPS receiver, a CAN interface towards the truck and an Ethernet interface towards the Interactor. The interfaces can be seen schematically in Figure 2.6. The C200 is connected to the yellow CAN-bus and has no interaction with the driver. The GPRS modem enables communication with the Fleet Management Portal (FMP), while the GPS receiver will make it possible to send the vehicle position to the FMP. The 200 contains a SIM card with a mobile phone number for access to the GSM network.

![Figure 2.5: The C200 “Blackbox” telematic unit.](image)

![Figure 2.6: Block scheme.](image)

The C200 has a wide range of applications that provide services like vehicle tracking and vehicle monitoring. In the future the C200 will be mounted in new Scania trucks and the owner of the vehicle will be offered a subscription to the services provided by the C200.

#### 2.9.1 C200 software architecture

This section is based on reference [20]

The C200 platform has a layered software concept as seen in Figure 2.7, where the low level layers handle all the hardware, while the applications are kept independent of the hardware. As operating system (OS), Linux is used. The OS provides support for IP-communication (IP Stack) by GPRS and Ethernet and offers IP-routing capabilities. Between the OS and the hardware a Board Support Package (BSP) is used. The BSP provides support code (drivers) to conform to the Operating system. The daemons are responsible for abstracting the hardware devices from the servers and applications. The server layer provides services to the application layer and is responsible for multi client connections.

![Figure 2.7: Layers](image)

At the application level numerous applications have been developed, each with its own functionality. All the applications at the application layer were developed using C++. With
A reference to Figure 2.8: An application is a single threaded framework with one capsule and a collection of sockets attached to it. The framework is common for all applications and schedules socket events. Communication between applications is handled by the concept of notifies and subscriptions. The data sent between applications has a parameter attached to it. Each parameter represents a specific kind of data. By subscribing to a parameter from another application the data will be received as soon as the data with that parameter is sensed on the socket. Subscriptions are only intended to work in bottom-up manner, i.e. a server layer application is not supposed to subscribe for parameters from an application at the application level. It is possible for an application at the Interactor layer to subscribe to parameters from the server layer.

To send data to an application notifies are used. A notify is a type of message that contains the data and the parameter associated with the data. The notify message is then sent to the intended application, attached at the input socket.

The applications also contain a timer socket with a timer object which calls a specific method in the capsule when the timer object expires.

The server socket and client socket are only used when parameters are being subscribed by another application. The server socket then creates a client socket when a remote process connects to the server port.

The framework contains the functions which are called when the application receives a notify message. Those are standard functions inherited by every application from the framework. To do something different than the framework when receiving a notify, any of those functions must be overloaded with a function in the application capsule.
3 Existing remote diagnosis system architecture

This chapter will give an introduction to the remote diagnosis concept. It will explain Scania’s current system for remote diagnosis and its architecture. The system goes under the project name STEP.

This chapter has two purposes:
- To introduce the reader to existing software and hardware concepts that can be reused in a new system.
- Serve as a system analysis to identify requirements for a new system.

Each system of STEP will be explained in a separate section. The final section (section 3.6) will explain the overall functionality.

3.1 Overview

The STEP system, as seen in Figure 3.1, features a Web interface where the user can trigger readouts and fetch data from the database. The data from the database can then be presented to the user through the web interface. In the web interface the user can e.g. schedule readouts, trigger readout, view statistics of DTCs and classify DTCs per ECU. In the Interactor an application for remote diagnosis and a diagnostic software called The Scania Communication Module (SCOMM) are installed.

![Figure 3.1: High level overview.](image-url)
3.2 The CAN-bus in a Scania vehicle

In a Scania vehicle there are several Electrical Control Units (ECUs). Each control unit controls different parts of the vehicle such as the brakes (BMS) or the climate control (ACC). The communication between the ECUs takes place on a CAN-bus. There are three different CAN-buses, namely green, yellow and red. The three buses are connected with a coordinator system (COO), which routes traffic to avoid unnecessary conversation between the buses. On the red bus the most critical control system like brakes and gearbox control are located. On the green bus the least critical ECUs are located. ECUs sending to and from the red bus have a low message ID and hence have always priority over other traffic. [10]

The Interactor (RTI) is located on the green bus and together with the RTG they form the FMS connector. The RTG receives messages from the CAN-bus and gateways them to the FMS system. The CAN-bus in a Scania vehicle can be seen in Figure 3.2.

![Figure 3.2: The CAN-bus connecting all ECUs in a Scania vehicle.](image)

3.3 Diagnostic protocols

To communicate with ECUs a higher level protocol is required. Here either Keyword Protocol 2000 (KWP2000)\(^5\) or Unified Diagnostic Services (UDS) is used (different ECUs use different protocol). The protocols offer the user application layer services for reading out Diagnostic Trouble Codes

---

\(^5\) When referring to a keyword, this could be either a UDS or KWP2000 command.
(DTC), resetting the device, reading ECU information, uploading and downloading files to the ECU and so on. The user is allowed to use different services depending on what kind of session and security access level is used.

UDS and KWP2000 are ISO defined standards which define the application layer, data link layer and the physical layer. These standards do not specify all details, but some are left out to the manufacturers to specify. Therefore Scania has created Scania implementation requirements which define manufacturer specific settings.

### 3.3.1 Message flow

[This section is based on reference [11]]

Communication with an ECU comes in the form of request and responses. Figure 3.3 shows a request and response message flow example. The client uses the application layer services to request diagnostic functions to be executed in one or more ECUs. The server, usually a function that is part of an ECU, uses the application layer services to send response data. The data provided by the requested diagnostic service, is sent back to the client.

A KWP2000 request or response consists of a diagnostic service identifier (SID) and parameters that belong to that service. The service identifier describes what actions to be taken by the ECU if it is a request. If it is a response the service identifier will either signal a positive or negative response. A negative response will be sent if the operation was unsuccessful or not completed in time. The parameters sent along with the service identifier describe what the service identifier should do. For example, SID number 0x10 tells the ECU to change its diagnostic session. The parameter sent along with the SID would then describe which diagnostic session to enter.

![Figure 3.3: Generic message flow example.](image)

Each manufacturer defines application layer timing parameters. All ECUs shall in the normal case respond to a request message within T₁=100 ms. This means that each ECU shall start sending its response message within T₁ after the request message has been correctly received.

When sending a request the response that matches that request has to arrive before sending the next request. This is referred to as synchronous communication.
3.3.2 Diagnostic sessions

This section is based on reference [11]

To control the level of diagnostic functionality provided by the ECU, diagnostic sessions are implemented in the session layer. Different diagnostic sessions select different ECU functionality and access to different memory locations.

There shall always be exactly one diagnostic session active in an ECU. If no other diagnostic session is started, then the default diagnostic session shall be running as long as the ECU is powered.

When starting communication with an ECU, typically the appropriate diagnostic session for the intended keyword command has to be entered. By default the standard session is running in the ECU. To be able to run keyword commands with a higher level of security a session with a higher level of access has to be entered.

3.4 The Interactor application

This section is based on reference [21]

The application is written in .NET remoting and has the components according to Figure 3.4. A short description of the data flow and the most important components will follow.

![Figure 3.4: Application components.](image)

When a settings file (a XML script for triggering readouts) arrives to the Comm Adapter, it is parsed by the Settings Parser. If the settings script specifies a scheduled readout, the information about start date and end date is sent to the scheduler. If the script defines an error code readout, it is handled by the Reader.

The Reader also reads responses from ECUs and sends commands that are triggered by the Scheduler. The readouts are sent to the XML generator, which creates a XML script and stores it on the Interactor file system.

The File Transfer component uploads DTC readout files and downloads setting files to and from a web service on the server side. This takes place according to Figure 3.5.
The File Transfer component calculates and transfers a MD5 checksum of the uploaded files. When an acknowledgement is received that the file and the checksum match, the uploaded files are moved to a different folder in the file system. Downloaded files are compared to a MD5 checksum and sent to the Settings Parser. The transfer can be triggered by either the scheduler or by a push command from the request web-interface (see section 3.6).

3.5 SCOMM

The SCOMM software is an essential component in the remote diagnosis system. SCOMM is an Application Programming Interface (API). The API comes in the form of a dynamic link library (DLL) that the application in the Interactor uses. SCOMM performs the following:

- Takes care of ECU specific knowledge.
- Creates a higher abstraction level of the KWP2000/UDS protocol.
- Presents data from/to ECUs in understandable units, like V or rpm.
- Offers a generic low level interface to the KWP2000 services.
- Supports UDS in the same way as KWP2000.
- Handles security levels in the KWP2000 protocol.
- Secures ”authority” to protected ECU data by protecting security access keys.
- Enables communication with many ECUs at the same time.
- Enables synchronously execution of functions.
- Logs communication.

3.5.1 Architecture

The Scania Communication application contains the components depicted in Figure 3.6. The SCOMM library and SSK component together form the SCOMM component.

The main component is the SCOMM library which contains all the functionality for ECU communication. The SCOMM library handles communication with the application through interfaces (see section 3.5.2) and the VCI2. The VCI2 is not part of the SCOMM application but

---

6 There are several other components not mentioned here. Those components are not essential to this project.
SCOMM is written to function with the VCI2. It converts USB serial data to CAN serial data and can thus be used to connect a computer to the CAN-bus using the USB port. SSK handles the HASP USB-key. The HASP USB-key ensures that the mechanic or engineer has the necessary permission to adjust and check the vehicles ECUs. Different USB-keys have different levels of security. With an engineer USB-key the user can access all services offered by SCOMM. [22]

![SCOMM Architecture](image)

**Figure 3.6: The SCOMM architecture.**

The HASP USB-key contains a crypto key used for the KWP2000 SecurityAccess service. Some services in the ECU are restricted by a security access. Before a user can access those services a crypto key exchange with the ECU has to take place. If the crypto key in the HASP USB-key matches the one in the ECU, the services corresponding to that security access are unlocked. [10]

### 3.5.2 Interfaces

*[This section is based on reference [13]]*

Interfaces are used for communication between an application and SCOMM. Pointers to these interfaces are fetched from SCOMM by the application. Interfaces can be represented as in Figure 3.7, where the product and the connected ECUs have an interface to the application. Different interfaces provide different kinds of services to the application. Some of the interfaces provided are:

- **Product interface** – The interface contains functions for connecting/disconnecting communications to the product and functions for finding out, which ECUs are contactable.
- **Interface for ECU identification data** – This is an interface to the read identification data that is stored in the ECUs.
- **Interface for the common ECU** - is an interface that exists for all contactable and identified ECUs.
- **Interface for a list of error** – Functions for extracting information from a list of error codes.
- **Interface for an error code** – Functions for fetching information about a single error code.

---

7 Hardware Against Software Piracy (HASP) is a hardware-based licensing and protection system [29].
3.6 Comprehensive functionality

With reference to Figure 3.8: When a user requests a readout or the sending of an individual keyword command in the DTC web-interface, an XML settings script of readout specific data (what to read out, how often and if any extra commands shall be send along) is sent to the vehicle web-interface through the DTC database. The web-interface fetches the phone number from the DTC database of the specific vehicle(s). The phone number(s) is forwarded to the request web-interface. The request web-interface sends an SMS message to the vehicle requesting the vehicle to fetch the settings script (a so called push) from the vehicle web-interface.

When the settings file is sent over the GPRS link, the telematic service forwards it to the application. The application interprets the message and uses SCOMM to request a readout from the specific ECU(s). The response is send back to the application which creates an XML script and sends it to the vehicle interface and later to the DTC database. The result is presented to the user on the DTC web interface.
3.7 Other systems for remote diagnosis

[This section is based on reference [8]]

Before the STEP system was introduced an application called CDev_GSM was used. This was the first generation remote diagnosis system at Scania and was introduced in 1997. The CDev application contacts the vehicle with a GSM modem and the user can choose an ECU at a time for readout.

For each ECU the user can choose what to be read, DTC, ECU identity or Freeze Frames. CDev will then generate a file for each type of readout and the user can save it manually to a folder on the file system. A file resulting from a DTC readout can be seen in Figure 3.9.

This system for remote diagnosis was inefficient and lengthy because of all the manual work with text files. Its successor STEP, therefore, uses a database for the storage of readouts.

---

8 A freeze frame is additional data that may be stored together with the DTC to give information about the environment conditions at the time of DTC storage.
4 An architecture for a remote diagnosis system

In this chapter, a proposed architecture for a remote diagnosis system based on the new telematic unit C200 will be presented. Some goals on the system have been identified by interviewing engineers at Scania. These goals are presented under section 4.1. The overall goal is to create an architecture that is more flexible than Scania's existing system for remote diagnosis.

The remote diagnosis system will consist of a C200 application, new functionality on the server side, usage of the Scania communication library component (SCOMM), a script for transferring diagnosis information and a user web tool. Each component will be explained under a separate headline.

4.1 Goals with a future system

4.1.1 Flexibility

The vision for the master project was to create an architecture for a new remote diagnosis system over the C200 telematic unit. The system is supposed to replace the old STEP system but have all the functionality (the architecture is at least supposed to support the functionality to be implemented) that the STEP system has. [13]

In the current STEP system, a lot of the functionality is located in the application in the Interactor. This becomes troublesome when new functionality is to be implemented, since every application in every Interactor has to be updated. If new functionality only has to be implemented on the land based server side this would make for a more flexible system. [14]

4.1.2 Security

Today's system is not very secure since the Interactor is connected to the CAN bus via a VCI2 which means that anyone can connect their laptop to the CAN-bus. Also because SCOMM is installed in the Interactor a HASP USB-key has to be inserted in the Interactor.

4.2 Overview of the system

The system will have a similar architecture as the one of STEP except that the Interactor has been replaced by a C200 telematic unit and the SCOMM software has been moved to the server side. The system is schematically shown in Figure 4.1.
To meet the goals with the new system the starting point has been to put the SCOMM component on the server side and to keep the C200 application as simple as possible. Putting SCOMM on the server side means that no VCI2 or HASP USB-key has to be used in the vehicle. Instead the C200 CAN-interface is used for communication with the CAN-bus. One can think of a system without the SCOMM component but this type of system has not been taken into consideration in this project.

The C200 application should be as simple as possible. It should be able to parse and emit scripts and to send and receive messages from both FMP and the CAN-bus. If compression is used the application has to support compression and decompression of the script file.

4.3 Communication between the FMP and the vehicle

Communication with the FMP is done via a communication server, which contains a communication server agent for posting messages to the out queue. Payload data can either be sent as a string or a byte vector. For sending scripts, an application which puts the script in a byte vector is needed. This can easily be done in .NET with System.text library and support for this is included in the C200 framework.

4.3.1 The communication protocol

Along with the C200 telematic unit a completely new message protocol for communication between the vehicle and the Internet server (FMP) will be introduced. The protocol is based on UDP and is designed to increase the data part in every package. The protocol does also feature an ARQ\(^9\) scheme for reliable data transfer on the lower levels.

The protocol does not provide a proof of delivery at the application level. Therefore, applications which need proof of delivery must provide an ACK/NACK scheme on the application level, sending them as payload messages. [15]

4.4 Suitable scripts

The proposed architecture is all based on the sending of scripts over the GPRS link. To create an efficient system several requests have to be bundled together rather than requests sent one at a time.

\(^9\) ARQ is an error control method for data transmission which uses acknowledgments and timeouts to achieve reliable data transmission.
This is most efficiently done by sending scripts that can be parsed at each end. The script should contain as less overhead as possible to increase the data part in each packet. This is because Scania is billed per megabyte of transferred data. Here, different scripts are presented and their advantages and disadvantages are listed.

4.4.1 XML

XML is the script used in the current remote diagnosis system, STEP.

The Extensible Markup Language (XML) is a general-purpose markup language\(^\text{10}\). Its primary purpose is to facilitate the sharing of structured data across different information systems, particularly via the Internet, and it is used both to encode documents and to serialize data. Where HTML does a very good job of marking documents for browsers, XML allows you to define any kind of document markup. This can, for example, be a document that describes a "to do" list for the C200 remote diagnosis application.

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<book collection>
  <book language="english">
    <title>Sherlock Holmes</title>
    <author>Arthur Conan Doyle</author>
  </book>
  <book language="swedish">
    <title>Röda rummet</title>
    <author>August Strindberg</author>
  </book>
</book collection>
```

Figure 4.2: A sample XML script.

Figure 4.2 shows a sample XML script. An XML script is built from elements. An elements looks like `<book>This is a book.... </book>`, where `<book>` is called a start-tag and `</book>` is called an end-tag. The script must contain exactly one root element. Here `<book collection>` is the root element. The root element can be preceded by an optional XML declaration. This element states which version of XML is in use. It may also contain information about character encoding and external dependencies. It is possible to add comments to the XML script by starting the line with `<!--` and ending it with `-->`.\(^\text{[30]}\)

4.4.1.1 Pros and cons of using XML scripts

Pros:

- XML scripts are an integrated part of .NET XML Web services and are therefore convenient to use.
- Support for XML scripts in .NET is extensive.
- XML scripts provide an easy-to-read hierarchical tree structure.
- There are a wide range of XML parsers available for all platforms.

\(^{10}\) A markup language is a set of annotations to text that describe how it is to be structured, laid out, or formatted.
Cons:
- XML scripts are verbose which results in large scripts [7].
- Parsing XML scripts demands both more CPU power and memory than other scripts which is not always available in embedded systems.

A way to make the scripts smaller is to use compression. Both .NET and the C++ standard library provide methods for compressing and decompressing streams of data. Reference [32] shows that compression ratios around ten times are possible. For XML scripts that contain small amounts of raw data even better compression ratios can be achieved. For more information on XML compression and a comparison between compression techniques see reference [32].

Traditional compression methods, however, offer only the advantage of compression, without the advantage of decreased parsing time. Using a binary XML format reduces the verbosity of XML documents and cost of parsing, but prevents the use of ordinary text editors to view and edit the document.

According to [31], most time of the parsing process is spent in the object-creation of building the XML tree structure. Thus finding better parsing techniques that reduce the object creation-cost is significant to XML parsing performance.

4.4.2 INI files

[This section is based on reference [33] ]

INI files are plain-text files that contain configuration information. These files are used by Windows and Windows-based applications to save information about preferences and operating environment. The name INI file comes from the filename extension that is usually “.ini”. INI stands for initialization.

An INI file is an 8-bit text file divided into sections, each containing zero or more keys. Each key contains zero or more values. Every key has a name and a value, delimited by an equals sign (=). The name appears to the left of the equals sign. An example can be seen in Figure 4.3.

```
[SectionName]
keyname=value
;comment
keyname=value, value, value ;comment
```

Figure 4.3: Example INI file.

Section names are enclosed in square brackets, and must begin at the beginning of a line. There is no explicit "end of section" delimiter; sections end at the next section declaration, or the end of the file. Sections may not be nested.

Semicolons (;) indicate the start of a comment. Comments continue to the end of the line. Everything after the semicolon is ignored. A Carriage return and a line feed marks the end of a statement. Section, key names and comments are case-insensitive and names cannot contain spacing characters.

Parsing INI files is an easy procedure because of the scripts’ simple structure. There exist open-source parsers which can be useful to look at, when building a parser.
4.4.2.1 Pros and cons of using INI files

Pros:
- INI files are less verbose than XML files.
- INI files have a simple structure which makes parsing less complex.

Cons:
- Not many INI file parsers are available.
- INI files are typically limited to two levels (sections and keys), thus nesting is not possible.
- INI files do not handle binary data very well.

4.4.3 JSON

[This section is based on reference [34]]

JSON (JavaScript Object Notation) is a lightweight data-interchange format. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages.

JSON is built on two structures:
- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures which in one form or another are supported in all programming languages.

JSON’s basic types are numbers, strings, booleans, arrays, objects and the null value. Figure 4.4 shows the JSON representation of an object that describes a person. The object has string fields for first name and last name, contains an object representing the person's address, and contains a list of phone numbers (an array).

```
{  
  "firstName": "Anders",
  "lastName": "Andersson"
  "info": {   
    "streetAddress": "Storgatan 1",
    "city": "Storstad",
    "state": "AB",
    "postalCode": 15100,
    "married": true
  },
  "phoneNumbers": [
    "01-0101010",
    "01-1234567"
  ]
}
```

Figure 4.4: Example of a JSON representation of a person.
4.4.3.1 Pros and cons for using JSON scripts

Pros:

- JSON is less verbose than XML.
- JSON supports nesting (you can place objects within other objects) which INI files do not.
- JSON has a wide variety of script emitters and parsers for the most common programming languages.

Cons:

- The support for JSON scripts in C++ is not as extensive as for XML. Most JSON emitters and parsers are not stand-alone, but require some additional library.

4.4.4 YAML

“YAML Ain't Markup Language” (abbreviated YAML) is a data serialization language designed to be human-friendly and work well with modern programming languages for common everyday tasks. YAML uses the fact that almost every data structure can be represented with three basic primitives: mappings (hashes or dictionaries), sequences (arrays or lists) and scalars (strings or numbers). This means that simple YAML files (e.g. key value pairs) are easily parsed with regular expressions without resort to a formal YAML parser. However using a language with support for dynamic data structure, e.g. Perl, Python or Ruby is favorable. [35]

Compared to XML, YAML is more closely targeted at data structures and messaging, where XML is designed to support a more structured documentation. YAML uses clean and very minimal structural indicators, relying largely on indentation of nested elements. This makes YAML a great deal simpler to read, edit, modify, and produce than XML. [36]

```
---
person:
  firstName:  Anders
  lastName:   Andersson
  profession: self-employed

workAddress:  &id001
  street:   |
    Storgatan 1
  city:     Storstad
  state:   AB
  postalCode: 15100

homeAddress:  *id001

phoneNumbers:
  - home:  01-0101010
  - cell:  01-1234567
...
```

Figure 4.5: Example of a YAML script representation of a person.

The example in Figure 4.5 defines a hash with four top level keys. The last key, "phoneNumbers", contains a two element array (or list). Relational data and redundancy removal are displayed: the "workAddress" hash content is copied from the "homeAddress" hash's content as indicated by the anchor (&) and reference (*) labels.
Multiple documents can exist in a single file/stream and are separated by "---". An optional "..." can be used at the end of a file for signaling an end in streamed communications without closing the pipe. [35]

4.4.4.1 Pros and cons of YAML scripts

Pros:
- YAML is very well documented.
- YAML is less verbose than XML.
- Supports nesting.

Cons:
- Bindings for the C/C++ language exist but their development is at a very early stage.

4.4.5 Summary and remarks

Both YAML and JSON are less verbose than XML and more flexible than INI files. Support for XML is more extensive than the other three though.

Not discussed in this section are binary file formats. Binary file formats can improve parsing performance and reduce verbosity of the scripts. The drawback is that binary formats hinder the use of ordinary text editors to view and edit the document [31]. Since there is no need to modify files in a remote diagnosis system, binary file formats, e.g. binary XML is an interesting solution.

4.5 How to utilize SCOMM

SCOMM is adventurous to use in this system because it will provide a wide range of services to the user. The interface in SCOMM gives the user a possibility to execute arbitrary KWP2000 or UDS commands to an ECU. SCOMM can also interpret hexadecimal valued KWP2000 responses into something useful like rpm or seconds.

There are basically two options to where SCOMM should be located. It can either be placed in the C200 as a Linux application or it can be installed as a windows application together with the remote diagnosis user interface. The former alternative is not applicable since SCOMM is strongly dependent on the Windows environment. Porting its code to a Linux platform is not an option. Also, using the SCOMM library in the vehicles becomes troublesome when new versions of the SCOMM library are released. That is because a technician has to visit every vehicle to update its software. Using SCOMM requires a HASP USB-key and a VCI which is not favorable to have in the vehicle.

Placing the SCOMM application on the remote diagnosis user computer will create a much more flexible structure. Hence all administration of the software is done centrally. Also, the remote diagnosis user computer will be running Microsoft Windows and SCOMM provides a setup file for simple installation.

There are also drawbacks involved with installing SCOMM on the server side. The main problem is that SCOMM is a complete system were SCOMM handles all layers in the OSI model. Because the C200 has a complete KWP/UDS stack implemented and will handle every layer from the application layer and down, which is not convenient. Somehow the application using the SCOMM library has to be separated from the application layer.
4.5.1 Adjustments to the SCOMM application

Adjustments must be made to SCOMM library software for it to fit in the architecture described in section 4.6. To avoid communication through the USB port, SCOMM can be set to operate in demo mode. This will save the communication from the SCOMM library to the application layer in a log file. This will require a simulation file of recorded diagnosis data, which is not favorable.

SCOMM has timing requirements for how long to wait for a response. Since according to [15], the GRPS link and the protocol used only guarantee that a packet will reach its destination and not when, these timing requirements have to be removed. Further the SCOMM library has to support asynchronous communication to be able to send several requests in one message. An extension that allows SCOMM to send several requests would be favorable.

4.5.2 Security issues

SCOMM provides a security system by using HASP USB crypto keys. Here different HASP keys give the user different level of access to the SCOMM library functionality. E.g. a key with a low level of security cannot access functions to reprogram an ECU. By using the SCOMM library with its crypto key system, unauthorized users can be prevented from using the system. When using SCOMM, a crypto key is needed even when executing keyword commands in the default session.

4.6 Conceivable architecture for the server side

One can think of many architectures for the server side. Here one of them is presented.

The architecture was prepared by looking at the current architecture for the FMP (reference [16]), the architecture of STEP (reference [17]) and adapting it to using the SCOMM library. Also the system goals in section 4.1 have been taken into consideration.

The architecture can be seen in Figure 4.6 and is composed of the following components:

- DTC web site – Here the user can trigger readouts and look at readouts.
- Message queues – Queues are used for communication between software components.
- SCOMM DLL – Provides security, interprets readouts and abstracts keyword commands. The DLL shall be included in the DTC Web site project.
- File system – Permanent memory for storing of diagnostic files.
- Windows service – Runs in the background and polls queues for messages. Outgoing messages are emitted into a script and compression is applied to reduce the scripts size. Incoming messages a decompressed and parsed.
- Web service – This is the façade\textsuperscript{11} to the FMP communication services. Contains functionality for sending messages to and from the Communication server.
- DTC database – Database for permanent storing of readouts.
- Communication server – Provides access to the GRPS network.

\textsuperscript{11} Object that provides a simplified interface to a larger body of code.
The message flow is in the following order:
1. Triggering of a readout is done from the DTC Web site with help from functions in the SCOMM DLL. The DTC Web site records the keyword traffic generated by SCOMM and a message is created from that traffic.
2. The message is sent to a Windows service through inter-process communication (IPC).
3. The Windows service creates a script, applies compression to it and sends it to a Web service and to the FMP.
4. The response is put in a message queue by the Web service.
5. The Windows service polls the queue for incoming messages. Incoming messages are decompressed and parsed. The message is parsed and saved in a database. The message is saved in a format that SCOMM can interpret.
6. The DTC Web site can be used to fetch information from the database to create statistics of error codes and additional statistical features. SCOMM is used to interpret the keyword commands stored in the database. The database can also be used by the C200 to fetch messages containing e.g. instructions for scheduled readouts.

For inter component communication messages queues can be used. An option to message queues can be to save files to the file system. This method can be used for communication from the Windows service and to the DTC Web site. This is an appropriate method because files stored on the file system can be loaded by the SCOMM library [13], but messages from message queues cannot be fetched.

Since SCOMM is located on the server side, keyword commands must be sent across the GSM network, which will generate much traffic. If SCOMM could be run in the C200 only instruction would be necessary to send across the link. This is the drawback of putting SCOMM on the server side.
4.6.1 Security issues

The proposed architecture is based on communication with XML Web services. Since the .NET Web services are publicly available there are security issues involved when using them. Also XML Web service security standards have yet to be agreed on and have to be widely adopted. An idea could be to use credentials to authorize against a XML Web service. It is also possible to use ASP.NET authorization controls to restrict access to web service resources [7]. For more information on web service security see reference [9].

4.7 Requirements on C200 application

The application is supposed to be as simple as possible to minimize the risk of updates done to it (see section 4.1.1). The most functionality is supposed to be at the server side. Figure 4.7 shows the location and dependencies of the RemoteDiagnosis application. The RemoteDiagnosis application subscribes to the remote diagnosis parameter defined in the Communication Handler and the parameter from the client part of the KeyWord Manager.

This section does not present a full software architecture since the application can be written in many ways depending on what services should be supported. Instead, functionality that the C200 application should provide is listed.

![Figure 4.7: C200 applications architecture.](image)

The application can receive and send messages to and from the KeyWord manager and the Communication Handler. The Communication Handler takes care of messages sent to and from the FMP. The KeyWord Manager takes care of messages to and from the CAN-bus.

The application (RemoteDiagnosis in Figure 4.7) should take care of the following:

- Decompress incoming diagnostic scripts from the Communication Handler.
- Parse the decompressed script into individual keyword requests or other commands.
- Send individual keyword requests to the KeyWord Manager or execute other commands.
- Receive incoming keyword responses from the KeyWord Manager.
- Emit a diagnostic script from the response.
- Compress the script and put it in a byte vector for transmission to FMP.
Send the script as a notify message to the Communication Handler.

The protocol Scania uses over the GPRS link only guarantees that the message will be delivered to the application layer. If an ECU is not addressable a response will not be returned. This is part of the application layer and is not handled by the Scania protocol. In this case the C200 application must send a failure message to FMP as soon as the keyword session times out.

In theory it should be possible to use all services provided by the SCOMM library. Some services should though be restricted for security purposes. This restriction should be handled by the server side.

4.7.1 Starting an ECU session

When the user starts the graphical user interface, knowledge of online ECUs is not available. To find out which ECUs are contactable an initial request-response message has to be sent over the link. The message contains a broadcast message on the CAN-bus asking all ECUs to enter their standard session. Online ECUs will then respond and a message with online ECUs can be sent to FMP. The graphical user interface can then display all online ECUs on the CAN-bus for the user. The message exchange can be seen in Figure 4.8.

![Figure 4.8: Setting up ECUs.](image-url)
5 Implementation of a demonstration system

One part of the thesis project was to implement a demonstration system that shows that it is possible to use the C200 for remote diagnosis. The system is not supposed to be used by others but merely to function as a demonstration of how to perform remote diagnosis over the C200 telematic unit.

The demonstration system is supposed to feature a user interface where arbitrary keyword commands can be executed against an ECU remotely. The system is also supposed to show how error code readouts could be implemented in the new system. The system goes under the project name RemoteDiagnosis. All new or modified components were built by the author of this thesis.

The demonstration system does not have all the features that a complete system would have. Instead several shortcuts have been taken to be able to finish the project on time. See section 5.4 for more information.

The demonstration system consists of two parts: components added on the server side and an application running on the C200. Each part of the system will be explained under separate headlines.

5.1 The server side

The server side consists of the Development computer and the Communication server. On the development computer a modified Web service and a user interface are running. The communication server is running services for adding messages to message queues, send messages over the GSM network and to route incoming messages.

5.1.1 The user interface

The user interface is a .NET Windows application written in C#. It has a web reference to the Web service (see section 5.1.3), a SCOMM log file script parser, functionality for printing results to the screen and a file system watcher, which monitors the file system for changes.

When triggering a readout of DTCs, an ECU must be selected from the drop down list seen in Figure 5.3. When clicking the “Read All DTCs” button the user interface creates an interface to the

![Diagram](image)
SCOMM product (1) and an interface to the ECU specified in the drop down list. The SCOMM library then saves the keyword commands to a log file. The log file is parsed by the Remote Diagnosis interface (2) to create a byte vector request message. The structure of the request byte vector can be seen in Figure 5.2. This type of structure for the message was chosen because its simplicity to emit and parse.

![Figure 5.2: Structure for the request message byte vector.](image)

The byte vector is sent (3) to the Web service. The response comes in form of a snapshot file (demo file) saved on the Development computer file system (4). The RemoteDiagnosis interface has an instance of FileSystemWatcher13 class. When the demo file is changed it is loaded into another interface of the SCOMM library (5). The SCOMM library interprets the demo file and the result is presented to the user (6) in the result window.

To use the SCOMM API in a .NET development environment a C# wrapper on top of SCOMM has been used. The wrapper enables the use of C# to create the SCOMM interfaces. The SCOMM functionality is more thoroughly described in section 5.1.2.

When an arbitrary keyword command is sent, the SCOMM interface is not used. Instead the RemoteDiagnosis interface creates a request message from the keyword command entered in the graphical user interface. The request message is sent to the Web service and to the C200.

---

12 The number within the parenthesis is in reference to Figure 5.1.
13 A class in the .NET Framework Class Library. It listens to the file system change notifications and raises events when a directory, or file in a directory, changes.
5.1.2 Using SCOMM

Since SCOMM is not designed for remote diagnosis purposes some adaptations had to be made. By setting the SCOMM library to run in demo mode it will save its communication to the application layer, in a file. This means that the information sent by SCOMM to the application layer services is saved to a log file instead of transmitting it on the USB port. Figure 5.4 shows an example of such a file.

Running SCOMM in demo mode also means that an actual ECU is not contacted, but instead the response is taken from a demo file. The demo file is actually a recorded simulation from communication with an ECU. For the SCOMM library to find a response from a request that response must be found in the demo file. This method is useful in a demonstration system but not effective in a real system.
When the response arrives, the Web services method will create a so called demo file (see Figure 5.5), which SCOMM can load. The demo file contains simulation data, which the SCOMM library uses to create a response when running in demo mode. In Figure 5.5, the byte code within the square brackets is the keyword request. The next line describes the response and the line after that is the response hex code.

By executing a readout one more time the responses in the demo file will be used. The response (error code) can then be interpreted by the SCOMM library into error code identifier, status, occurrence count and a timestamp when the error code was created. The DTC number or error code identifier is a two-byte hexadecimal number in the range 0001<sub>hex</sub> - FF<sub>hex</sub> and is unique in every ECU. The DTC number can be used by a mechanic to find a faulty part. The occurrence count is the number of times the specific DTC has been generated. The time stamp is the date and time for the last occurrence of the specific DTC [23].

Figure 5.4: A SCOMM communication log.
5.1.3 The Web service

To be able to post and receive messages to and from the communication server a Web service has been installed. It is written in .NET and contains functionality for sending messages to and from the FMP communication server.

The Web service contains a web method which takes a byte array as argument (see number 1 in Figure 5.6). The web method then creates a message with a unique type code and attaches the telephone number to the message. The message is then sent to the communication server (2).

The Web service also contains a method for receiving messages from the communication server (3). The input is an XML string where the data part is extracted. The data is used to create a demo file which is saved to the file system (4).

The Web service was installed on the development computer to be able to save files to the local drive. Another solution could be to install the Web service on a dedicated web server and save demo files to a network drive.

![Figure 5.5: A SCOMM demo file.](image)

![Figure 5.6: The Web service.](image)
The most important component in the Web service is the utility DLL for sending messages. The utility DLL contains functions for creating valid messages on the FMP and to post them to a queue. This is the only reference the Web service needs to have.

### 5.1.4 The communication server

To access the C200 in the vehicle the Fleet Management Portal (FMP) service is used. The only part of the FMP used for this project is the Communication server. The infrastructure of the Communication server cannot be changed, but every new message needs new components for routing of the message.

With reference to Figure 5.7: The communication server contains message queues for messages out (Out Queue in Figure 5.7) from the system and in (In Queue) to the system. The queues are regularly polled for new messages to send.

The Scania IP server establishes contact with APN for transfer of the packet to the C200. When the communication server receives a message from the APN it is put in the queue. The message routing table has been modified to route the RemoteDiagnosis message type to a message handler COM+ component. The message handler forwards the message to a .NET message owner COM+ component which forwards the messages to the Web service on the Development computer.

![Figure 5.7: The communication server.](image)

### 5.2 The C200 application

The RemoteDiagnosis application basically forwards messages from the Communication Handler to the KeyWord Manager and vice versa. Changes have been done to the Communication Handler for the C200 to recognize a RemoteDiagnosis message type.

![Figure 5.8: Request – response scheme.](image)

14 Access Point Name or APN is an access point for GPRS.
With reference to Figure 5.8: A KeyWord request message is a byte array with a unique parameter that the RemoteDiagnosis application subscribes to. The byte array can contain one or several KeyWord requests with a structure seen in Figure 5.2.

The RemoteDiagnosis application parses the byte array and sends individual KeyWord requests to the KeyWord Manager. Before the requests are sent to the KeyWord Manager they are saved in a buffer. The buffer is used to attach the request with its corresponding response. The KeyWord response message will then look as depicted in Figure 5.. The KeyWord response message is then sent to the Communication server through the Communication Handler.

The RemoteDiagnosis binary is small (30KB) and is only active for a short period at a time. The most time spent in the application is when parsing and emitting messages.
5.3 Overview of the complete system

Figure 5.10 shows the architecture for the complete system.

The main benefits of this system are:

- It does not require a HASP USB-key in the vehicle, because SCOMM is located on the server side.
- It does not need a VCI in the vehicle, because the C200 CAN interface is used.
- It uses the same protocol and server infrastructure as all other C200 services.

5.4 Limitations

To finish the project on time and because the C200 is not fully developed, the demonstration system has the following limitations:

- It is only possible to address one ECU at a time. If another ECU is addressed it has to be sent in another packet. This is a limitation in the user interface.
- It is only possible to address one C200 at a time. The protocol supports multiple recipients but the user interface is not configured to receive responses neither send requests to several C200.
- Since it is not possible to broadcast on the CAN bus, the ECU address has to be known in advance. The feature to broadcast on the CAN-bus is currently not implemented in the C200.
- When sending arbitrary keyword commands it is up to the user to make sure the keyword is not doing something destructive or unintentional.
- The demonstration system does not use compression of scripts.
• To run the demonstration system on another computer then the authors’ the communication server routing table has to be changed to point at another computer.
• It is only possible to send one keyword request at a time when sending arbitrary keyword commands.
6 Conclusions and recommendations

An architecture for remote diagnosis has been proposed. The architecture emerged from building a demonstration system which can execute arbitrary keyword commands and perform error code readouts from a user interface. Ideas for the architecture also emerged by looking at the current system for remote diagnosis and by interviews with its creators. The current system contains several features that could be implemented in the new system. Their implementation specific details have not been investigated, but an architecture that supports the implementation of such features has been proposed.

The demonstration system uses the same server infrastructure and protocols as other C200 services, which is favorable. The system is solely for demonstration purposes. Nevertheless the system gives an idea of how a remote diagnosis system can look like.

The benefit of using SCOMM is that it provides security by using crypto keys. If SCOMM is not used some other kind of security system has to be used. In this case the security access crypto key has to be sent in the script, which has not been investigated. A way to add security is by limiting access to the web service. This could be done by using credentials when accessing the web services or to restrict access to the web service for specific IP addresses.

If a simple diagnosis keyword is to be sent, it is sufficient to send a byte vector instead of a script file. This is the solution in the demonstration system where no additional information other than byte codes where sent across the link. If, for instance, chassis numbers of the vehicle or timestamp was to be sent across the link this would require for a script file.

Several script solutions have been presented in section 4.4. Support for XML script is extensive and by using a binary file format the drawbacks of XML scripts can be compensated. This is an interesting script solution that would be useful in a remote diagnosis system.

The most interesting script according to the author is JSON. It has all the features that a diagnostic script would require. Parsing and emitting this kind of script should not be a problem either. The transmission of script can become more efficient if the script is compressed at each end. By using an already lightweight script, e.g. JSON together with compression would be the best method for a script solution.
7 Future work

This thesis work does not investigate the security of the system at its fullest. A more thorough analysis of the different security aspects should be done.

The implementation of an extension to SCOMM which can send asynchronous requests without waiting for a response (as mentioned in section 4.5.1) should be done. The solution in the demonstration system is not satisfying. It relies on recorded simulation files from a truck and if a specific keyword command does not exist in the simulation file, an error occurs. Since new functionality is added to the ECUs continuously, working with simulation files will become troublesome. A way to avoid simulations files should be implemented.

A vision the author has had is that remote diagnosis could be an integrated part of the FMP. Diagnosis messages could be published on the FMP just as any position or vehicle monitor message. A mechanic could then log in to the FMP and trigger a readout and read the result. This would require planning for new functionality on the FMP, which is beyond the scope of this master project.
References

Literature

Specifications

Scania internal documents
[14] REI07329.doc – Master project definition
[16] Software Architecture Document – FMP 7.5 Message platform 2.0.doc
[17] Miljö nuvarande lösning.doc – Current remote diagnosis system server side architecture
[18] Software Architecture Document SVIP.doc
[19] Project_def_C200_MECO_384802_1.2.doc – C200 Project definition
[20] C200 framework.ppt
[21] 1533-002-001SS04.pdf – Interactor remote diagnosis application
[22] SCOMM presentation.ppt
[23] TB 1526.doc – Diagnostic Functions in Scania ECUs

Internet
http://web.archive.org/web/20050310133243/http://asd-
www.larc.nasa.gov/barkstrom/public/The_Standard_Waterfall_Model_For_Systems_Development.htm


[27] Microsoft Corporation, *Queues in Windows Communication Foundation*

http://msdn2.microsoft.com/en-us/library/x05s00wz(VS.80).aspx

http://www.aladdin.com/hasp/default.aspx

http://www.w3.org/TR/xml

[31] Jimmy Zhang (August 2006), *The Performance Woe of Binary XML*
http://webservices.sys-con.com/read/250512.htm

[32] Department of Computer Science United States Air Force Academy (September 2007), *An Analysis of XML Compression Efficiency*

[33] *What Is an .INI File?*
http://my.execpc.com/~iniman/what_is.html

[34] *Introducing JSON*
http://json.org

http://www.yaml.org/spec/current.html

[36] David Mertz, Gnosis Software, Inc. (October 2003), *YAML improves on XML*
# Appendix A - Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>Acknowledge</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>APN</td>
<td>Access Point Name</td>
</tr>
<tr>
<td>ARQ</td>
<td>Automatic Repeat-Query</td>
</tr>
<tr>
<td>BSP</td>
<td>Board Support Package</td>
</tr>
<tr>
<td>C200</td>
<td>Communicator 200</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>DISCO</td>
<td>DISCOvery</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Link Library</td>
</tr>
<tr>
<td>DMZ</td>
<td>DeMilitarized Zone</td>
</tr>
<tr>
<td>DTC</td>
<td>Diagnostic Trouble Code</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic Control Unit</td>
</tr>
<tr>
<td>FMP</td>
<td>Fleet Management Portal</td>
</tr>
<tr>
<td>FMS</td>
<td>Fleet Management System</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GNU</td>
<td>GNU's not UNIX</td>
</tr>
<tr>
<td>GPRS</td>
<td>General Packet Radio Service</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile Communications</td>
</tr>
<tr>
<td>HASP</td>
<td>Hardware Against Software Piracy</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines corporation</td>
</tr>
<tr>
<td>INI</td>
<td>Initialization</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter Process Communication</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>KWP2000</td>
<td>Keyword Protocol 200</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MD5</td>
<td>Message-Digest algorithm 5</td>
</tr>
<tr>
<td>MSMQ</td>
<td>Microsoft Message Queueing</td>
</tr>
<tr>
<td>NACK</td>
<td>Negative ACKnowledge</td>
</tr>
<tr>
<td>OBD</td>
<td>On-board Diagnostics</td>
</tr>
<tr>
<td>OSI</td>
<td>Open System Interconnection</td>
</tr>
<tr>
<td>SAE</td>
<td>Society of Automotive Engineers</td>
</tr>
<tr>
<td>SCOMM</td>
<td>Scania COMunication Module</td>
</tr>
<tr>
<td>SIM</td>
<td>Subscriber Identity Module</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure SHell</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TDMA</td>
<td>Time Division Multiple Access</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UDS</td>
<td>Unified Diagnostics Services</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>VCI2</td>
<td>Vehicle Connection Interface 2</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>YAML</td>
<td>YAML Ain't a Markup Language</td>
</tr>
</tbody>
</table>
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© [Anders Björkman]