MASTER'S THESIS | LUND UNIVERSITY 2016

Event based diagnostics on heavy duty vehicles

Marcus Birksjö, Johan Winér

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2016-44

Lunds Tekniska Högskola
Institutionen för datavetenskap

Examensarbete

Event based diagnostics on heavy duty vehicles

by

Marcus Birksjö, Johan Winér

2016-06-18

Handledare: Sven Gestegård Robertz, `sven.robertz@cs.lth.se`
Examinator: Björn Regnell, `Björn.Regnell@cs.lth.se`

Abstract

The integration of small computer units in vehicles has made more complex in-vehicle functionality possible. To troubleshoot these more advanced functions, the diagnostic services need to read out more data from the integrated computer units, causing an increased network load. This has caused a demand on more efficient ways of performing vehicle diagnostics.

One possible way to decrease the network load is to use an event based service. The aim of this thesis was to investigate the benefit of an event based service within the domain of vehicle diagnostics as well as to present a recommendation of how such a service should be designed. The thesis also aimed at eliciting obstacles and pitfalls connected with the implementation of the service in the current software architecture in heavy duty vehicles.

An industrial case study was performed at the Swedish company Scania to elicit the advantages, problems and limitations with an event based service for vehicle diagnostics. First a set of experts representing different domains within vehicle diagnostics were interviewed to investigate the need for an event based service. Requirements were elicited and compared with the event based service ResponseOnEvent defined in the ISO standard 14229-1:2013. A decision was then made to diverge from the standard in order to increase the number of fulfilled requirements and flexibility of the service. A new proprietary service was thus created and evaluated through a proof of concept implementation where a prototype of the service was implemented in two control unit.

The prototype implementation of the proprietary service highlighted multiple difficulties connected with the realization of an event based service in the current software architecture. One of the biggest problems was the fact that diagnostic services was assumed to always have a one-to-one relation between request and response, which an event based service would not have. Different workarounds were discovered and assessed. Another problem was the linking between an event triggered response message and the trigger condition. It was concluded that some restrictions would have to be made to facilitate the process of linking a response to its trigger condition. Non-determinism was another problem, since there were no guarantees that an event would not occur too often causing a network overload. In the final recommendation there are suggestions of how to solve these problems and some suggested areas for further research.

The thesis argues that there is a need for a new way to diagnose vehicle functionality due to their increased complexity and the limited bandwidth of today's in-vehicle networks. The event based service ResponseOnEvent offers a good alternative but might lack some key functionality required by an event based service. Therefore it is valuable to consider a proprietary service instead to maximize the benefits of an event based service. Due to its nature, an event based service might require a restructuring of the system architecture and limitations in the hardware might limit the usability and flexibility of the service.

Keywords: Event based service, Response on Event, ECU, Vehicle Diagnostics, UDS, KWP.

Sammanfattning

Integrationen av datorenheter i dagens fordon har gjort nya och mer avancerade funktioner möjliga. För att kunna verifiera att dessa funktioner fungerar och för att felsöka dem så används så kallade diagnostjänster. Allt eftersom funktionerna i fordon blir mer avancerade och beroende på data från flera olika sensorer och datorenheter så ställs nya krav på diagnostjänsterna att kunna läsa ut mer data ur datorenheterna. Detta orsakar en ökad nätverkslast och eftersom bandbredden på nätverket är begränsad ser man idag ett behov av nya diagnostjänster som kan utföra diagnos på ett sätt som ger upphov till mindre nätverkslast.

Målet med denna studie var att undersöka om en eventtjänst kan underlätta diagnostiken av fordonsfunktioner inom olika områden och vilka problem och begränsningar som finns. En fallstudie gjordes på den svenska motor och fordonstillverkaren Scania för att undersöka vilka områden som skulle kunna tänkas ha nytta av en eventtjänst och vilka krav de ställde på tjänsten.

Kraven som erhölls jämfördes med eventtjänsten ResponseOnEvent som är definierad i ISO standard 14229-1:2013. ResponseOnEvent visade sig inte kunna uppfylla alla de krav som ställdes på en eventtjänst och därför designades en proprietär tjänst som ett alternativ. En prototyp av den proprietära tjänsten implementerades i två av Scanias styrenheter för att undersöka problem och begränsningar i samband med en realisering av tjänsten. Ett av de största problemen som sågs var det faktum att den befintliga arkitekturen inte hade stöd för att skicka diagnosmeddelanden per event, något som en eventtjänst skulle kräva. En omstrukturering av den befintliga mjukvaruarkitekturen skulle krävas. Ett annat problem var kopplingen mellan ett eventtriggat meddelande och själva eventet. Icke-determinism var ett annat fundamentalt problem med en eventtjänst. Ett event skulle kunna ge upphov till skickandet av flera meddelanden vilket momentant skulle kunna överbelasta nätverket, det är därför viktigt att vidta åtgärder för att hindra ett sådant beteende.

Uppsatsen bekräftar ett behov av ett nytt sätt att diagnostisera funktioner i fordon och att en eventtjänst kan bidra till att minska nätverkslasten och erbjuda nya sätt att utföra diagnos på. ResponseOnEvent är ett bra alternativ men kan sakna några efterfrågade funktionaliteter som gör det värt att överväga andra alternativ. På grund av sin utformning kan en eventtjänst komma att kräva en omstrukturering av den befintliga mjukvaruarkitekturen och begränsningar i form av lagringsutrymme och beräkningskraft kan begränsa tjänstens tillgänglighet.

Nyckelord: Eventtjänst, ResponseOnEvent, ECU, Fordonsdiagnostik, UDS, KWP.

Acknowledgments

We would like to thank all the experts we have been in contact with at Scania during the thesis, for taking their time to share their knowledge with us. We would also like to extend a special thank to our supervisor Andreas Jonasson at Scania for his support and guidance during the thesis. We also want to thank our examiners and supervisors at LTH and The University of Linköping for their help and feedback.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

The automobile was invented for more than 200 years ago. Since then a lot has happened. Due to the continuous evolution of technology and competition between different vehicle manufacturers, there has always been a strive to provide more features and advanced functionality in vehicles. In recent years much new functionality have been made possible due to the integration of electronic control units (ECU) in the vehicles. The number of ECUs in vehicles today can vary a lot between different brands and models. Some top of the line models contain over 100 ECUs. This makes a vehicle one of today's most advanced systems that an ordinary person owns. [19]

The ECUs in vehicles are connected to each other through a communication bus. Using this bus the ECUs can exchange information with each other using services defined by different communication protocols. The communication also makes it possible to troubleshoot the vehicle's electrical system and monitor the internal state of ECUs using diagnostic services defined in a diagnostic protocol. This has become a very important tool when it comes to vehicle maintenance and development of new functionality.

1.2 Motivation

The electrical systems in vehicles have kept growing since the introduction of ECUs. Today, vehicles' electrical system can consist of a handful of ECUs up to over a hundred. As an example the Mercedes S-class uses 72 ECUs spread over seven different communication buses [17] while a Scania truck has about 19 ECUs [36]. There are also other types of vehicles which do not contain as many ECUs, for example some of Volvo's wheel loaders which contains only four to six ECUs [47].

As electrical systems in vehicles get more advanced and the number of ECUs in them increases, new demands are being placed on the diagnostic services. To facilitate the process of troubleshooting the more complex functionality in vehicles the diagnostic services need to read out more data from the different ECUs. This causes an increased load on the network bus. The bus load is already high and a need for more efficient diagnostic services has therefore been seen. The new and more advanced functionality also calls for new ways of performing diagnostics.

There are different alternatives to reduce the network load. The thesis evaluates a service where an ECU can be configured to send notifications containing diagnostic information upon a given event, a so-called event based service. If this service could be implemented, it could possibly reduce the load on the network that the diagnostic services give rise to and enable new efficient ways of performing diagnostics which in turn could enable more advanced functionality and services.

The transport industry is a market with a low profit margin and high fees in case a driver can not deliver on time. Together with high costs for wreckers to carry away the vehicle in case of a break down, this makes it very important that a truck driver can trust that his or her vehicle will be able to complete a delivery job without any delays or breakdowns. To help drivers, Scania and many other companies have help desks to which a driver can call if he is having trouble with his truck. To improve the work of the help desk it would be beneficial if they could monitor the truck's internal status remotely in real time. This would however require that the whole truck is continuously scanned for changes which would generate a much higher network load than what the current network can support. An event based service could possibly perform the same service while creating a lot less network load. This is just one example of how an event based service could make new functionality possible in heavy duty vehicles.

The thesis was carried out on behalf of Scania which is a leading manufacturer of heavy trucks and buses as well as marine and industrial engines. Scania also provides and sells a wide range of service related products and financial services which is based upon diagnostic data from the vehicles.[29]

1.3 Aim

The aim of the thesis was to build knowledge concerning an event based service and the advantages and disadvantages associated with it. The aim was also to bring understanding about how such a service should be implemented in order to satisfy the potential domains within the area of vehicle diagnostics. Therefore different domains that might benefit from an event based service was investigated and what implications those would have on the implementation of the service. The service ResponseOnEvent described in the ISO-standard 14229-1:2013 [22] was investigated as a potential event based service to be used. It contained some parts open for interpretation which was interpreted and clarified. As a way to detect eventual pitfalls and obstacles with an event based service, a prototype was developed.

1.4 Research questions

To guide the project a set of research questions were written down. The aim was to find answers to these questions during the course of the project.

1. How does ResponseOnEvent defined by the UDS-standard meet existing needs of diagnostics on heavy duty vehicles?
2. How can an event based service be realized in order to meet these requirements?
3. Are there any problems connected to an implementation of the service?

1.5 Limitations

The investigation that was carried out during the thesis was limited to the embedded systems within heavy duty vehicles provided by Scania. The communication network was therefore Controller Area Network (CAN) and the ECUs used was a RTC (Road Traffic Communicator) also referred to as C300 and a TMS (Transmission Management System). There are a lot of other ECUs in Scania's vehicles with different hardware and software and it is desirable that the event based service should be able to run on any of them. Due to limited time resources only the C300 and the TMS were used for the implementation and testing and only a superficial study was conducted looking into the possibilities of using the service in other ECUs.

There might be many different domains within heavy duty vehicle diagnostics that can benefit from an event based service and it would be beneficial if all of them could have been taken into consideration. However it would have been too time consuming identifying all the possible domains and therefore only a few domains were looked into due to their previous indications of a need for a new way to diagnose vehicles.

Some problems concerning the implementation of an event based service were discovered during the thesis. Some of them were due to the software architecture at Scania and therefore these problems might not have been found if the thesis had been carried out at another company. It is also likely that if the thesis had been carried out at a different company other problems might have been found.

1.6 Disposition

The report is divided into five parts. In chapter 2 a theoretical background will be given explaining some of the basic knowledge related to the scope of the thesis. In chapter 3 the method used during the thesis is introduced followed by chapter 4 containing the results. Chapter 5 evaluates the findings and the method applied during the thesis as well as suggests some areas for future research. Chapter 6 summarizes the conclusions drawn from the study.

1.7 Division of the Work

The work presented in this report has been equally divided between its two authors. Johan has been working on the TMS, implementing the server application of the service and Marcus has been working on the C300 implementing the client application. Both authors carried out the interviews where Marcus focused on analyzing the transcripts and Johan formulated and listed the requirements. Johan was the one responsible for contacting and arranging the meetings with the interviewee. The interpretation of the service ROE was done by Johan as well as carrying out the testing of the server application in one of Scania's trucks.

1.8 Definitions

Many area specific terms and abbreviations are used throughout the report. Table 1.1 explains the most frequently used. Hexadecimal and binary values are sometimes used and they follow the convention that hexadecimal values are prefixed with 0x and binary values are written inside single quotes. The hexadecimal value 1 (one) is therefore written as 0x01 while the binary value of one is written '1'.

Terms	**Meaning**
ADAS	Advanced Driver Assistance Systems: Advanced functionality that helps the driver.
C300	A specific ECU on Scania's trucks.
CAN	Controller Area Network
ComP	Common Platform, ECU software platform developed within Scania.
Downtime	Time during which the vehicle cannot be used.
DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit
Event based service	Service that takes certain actions at certain events.
Event logic	Combination of event and action to be taken.
ISO	International Organisation for Standardization.
KWP	Key Word Protocol 2000
Operational Data	Data describing how the vehicle has been driven.
ROE	ResponseOnEvent, an event based service.
RTC	Road Traffic Communicator: ECU that handles the communication with off-board services.
TMS	Transmission Management System: ECU that handles change of gear.
Uptime	Time during which the vehicle can serve its purpose.

Table 1.1: Frequently used abbreviations and their meanings.

2 Theoretical Background

This section presents theory relevant for understanding the final results of the thesis. First a short introduction to vehicle diagnostics and the vehicle electrical system will be given. Then follows an introduction to the concept of electrical control units (ECU) together with the two ECUs used in the thesis. A short introduction to diagnostic trouble codes will also be given due to its central role in vehicle diagnostics. To understand how the communication between control units work using the Controller Area Network (CAN), the architecture of CAN will be described together with two diagnostic protocols. The domains chosen to be the focus of the thesis will also be introduced. Finally the event based service ResponseOnEvent will be presented in short if the reader would not have access to the ISO-standard 14229-1:2013 where the service is described in full.

2.1 Vehicle diagnostics

In today's vehicles there is a wide variety of functionality beyond the one that directly addresses the driver. For example, functionality such as the collection of statistical data or functionality used by mechanics to support maintenance and service of the vehicle. These functionality are all referred to as diagnostic functions since they facilitate diagnostics in different ways. In this section a brief introduction will be given to how diagnostic functions can be used for creating value for the vehicle manufacturer and their customers.

During the production of vehicles, diagnostic functions are used to parametrize the ECUs and verify that the vehicle is working correctly. Later when proprietary extensions are built onto the vehicle, such as cranes and flatbeds, diagnostics functions are used for configuring the interface to the extension. Diagnostics is also used continuously throughout the life time of the vehicle for repair and maintenance. Workshops can for instance order extraction of operational data and trouble codes from a vehicle before it visits the workshop. This can reduce the time that the vehicle needs to spend in the workshop. The workshop might also use the data to preorder spare parts for the vehicle. [7][32]

Diagnostic functions can also be used for creating services which can be provided to the users of the vehicles, such as fleet management. Through a fleet management system a manager can monitor a fleet of vehicles and coordinate their work and driving routes. The manager can also monitor vehicle characteristics such as fuel consumption. Another possible service is assistance from the truck manufacturer or third party which can relieve the truck in case of a sudden breakdown. The assistance service can diagnose the truck remotely by for example reading its trouble codes and give recommendations to the driver how to fix a problem or send an assistance car to deliver spare parts. [7] [23]

Diagnostic functions are also a useful tool during the development of new vehicles and vehicle functionality. By logging network data and reading error codes in the vehicle, the new functionality can be tested and its behaviour verified. Diagnostic functions also deal with retrieving operational data which can be used during the development of new functionality. The data can provide insight into how the vehicles are being used and what demands on new functionality that might give rise to. Reading operational data is also useful for recommending new vehicles to customers. By looking at how they have been using their trucks, a seller can recommend a truck optimized for the customer's needs. [32]

Other functionality that is supported through diagnostic functions is the extraction of data from the tachograph. A tachograph is a computer log of how the driver has been resting and driving. Another application area is extraction of data concerning emission levels. Both the emission levels and the number of hours a driver is allowed to work before the driver has to take a break are regulated by law and therefore it is important that such data can be accessed by the organisations that enforce the laws [38]. All the domains within the diagnostic area mentioned above are summarized in figure 2.1. [9] [11] [23]

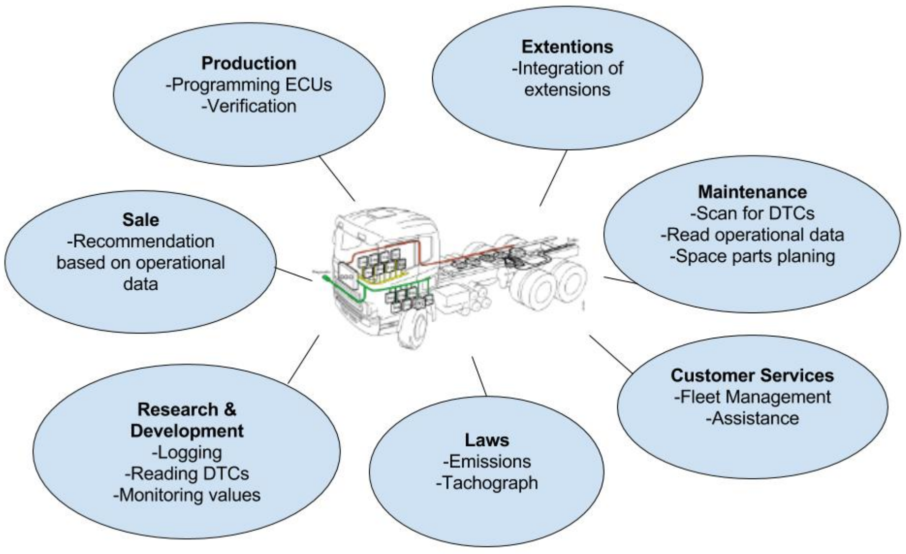

Figure 2.1: Usage of diagnostic functions

2.2 Electronic Control Unit

An electronic control unit (ECU) is a controller that can control one or several systems called actuators in a vehicle. The management of the actuators is enabled by the ECU's ability of reading values from a multitude of sensors as well as interpreting messages that reach the ECU through a communication medium such as the CAN bus [18]. The ECU can also send messages using the CAN bus. Examples of functions performed by ECUs in vehicles are ignition timing and shifting of gears in automatic transmission. In figure 2.2 a schematic figure of an ECU and its components are shown. [11] [13][23]

Sensors : Monitor and report values from their operating environment to the ECU. A sensor translates the working surrounding or a position into an electrical signal that can be interpreted and processed by the ECU.

Actuators: An actuator is a device that controls other mechanical or electrical devices. It translates electrical signals sent from the ECU into mechanical, hydraulic or electrical work.

Comm Receiver: Communication Receiver is a device that is connected to an internal network enabling communication between all of the ECUs connected to the network. Incoming signals to the communication receiver are translated into digital signals that can be processed by the ECU.

Comm Transmitter: Communication Transmitter is the transmitter of the processed information from the ECU. It transforms the digital information that the ECU wants to send into a physical signal which the transmitter then sends on the connected medium.

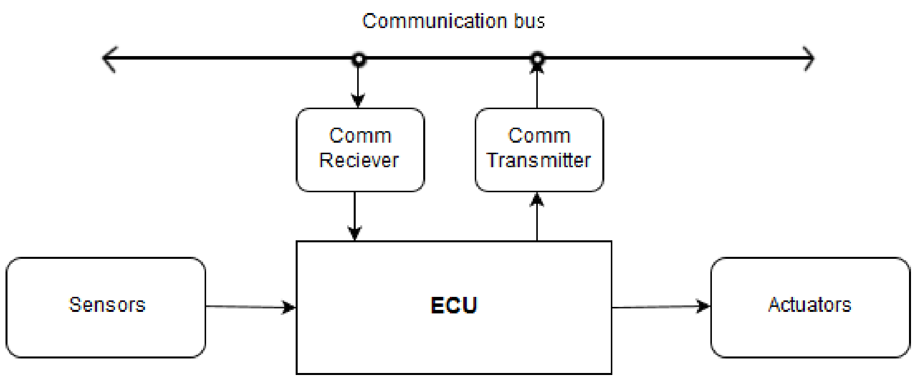

Figure 2.2: Electronic control Unit and its surrounding systems

2.2.1 Sessions

Sessions are a set of states which an ECU can be in. Each session defines a set of diagnostic services that it allows. The sessions are used to prevent an ECU from execute certain tasks that could be hazardous or in other ways undesired under certain circumstances, for example such as overwriting a certain value in the ECU or reprograming the ECU entirely while driving.

Each ECU starts in the default session and can then transition to another session like the extended diagnostic session or the programming session upon a request from another unit on the network. Sometimes certain conditions must be met before a certain session can be entered or before certain services can be used. For example an exchange of passwords needs to be done to get access to certain services. This is called security access. The passwords can be stored inside the vehicle but sometimes they are stored remotely in a server in which case the vehicle first needs to establish a connection to the server before it can complete the security access. [21][43]

2.2.2 TMS

The TMS is one of the ECUs which were used in the thesis and a short description is therefore required. The TMS was used as the server ECU for the prototype application and the reason why the TMS was chosen was since it uses Scania's software platform and still has some unused memory and processing power compared to some other ECUs where there are almost no free memory and processing power left. The fact that is uses Scania's software platform was important since any future implementation of the service would be done on this platform.

The TMS is located on one of the main buses in the communication network of Scania's trucks, see figure 2.6. It executes gear changes on vehicles with the Opticruise service (a service for changing gears). The driver communicates with the TMS through the brake pedal and accelerator pedal which are connected to the ECU through CAN. In figure 2.3 the TMS used in the thesis is shown. [27]

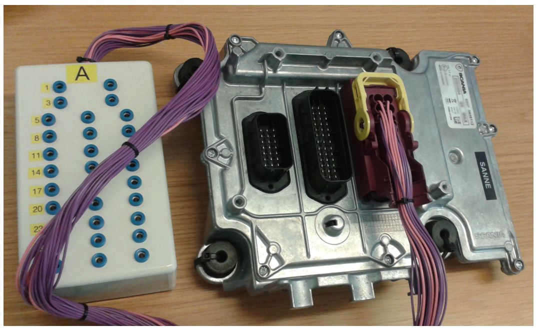

Figure 2.3: Photo of the TMS used in the thesis

2.2.3 C300

The C300 is the other ECU which were used in the prototype. The reason why the C300 was chosen was since it is used to communicate information to off-board applications from on-board systems by carrying out diagnostic services [28]. Diagnostics can thus be performed by an off-board system via the C300. A picture of the C300 can be seen in figure 2.5.

The C300 is located on one of the main buses and has the ability to send diagnostics request messages using either the KWP or UDS protocols. The functionality of the C300 is divided into applications. Each application can be seen as a separate service which can send and receive diagnostic messages from the communication network using the Diagnostic Manager. The different layers of the C300's software can be seen in figure 2.4.

Diagnostic Manager: There can be several applications within the C300 sending and recieving diagnostic messages. However the C300 can not handle multiple request simultaneously since it needs to be able to map the incomming response message to the request message. The diagnostic manager solves this by only allowing one application at a time to send messages on the bus. Once the response message has been received and mapped to the request message the bus is free again and the next application can use it.

CAN Server: Handles the CAN bus, receives incoming messages and transmits outgoing messages.

System Manager: All applications are monitored by the system manager. It starts the applications in a specified order and makes sure that they are all running as they should.

Platform Gateway: Handles communication between the higher and lower layers. The communication is based on overloaded functions and call-back functions which are provided by the platform gateway.

ResponseOnEvent: Is the application intended to enable the client functionality of an event based service. This application shall be able to set up, start, stop and handle incoming response messages from other units.

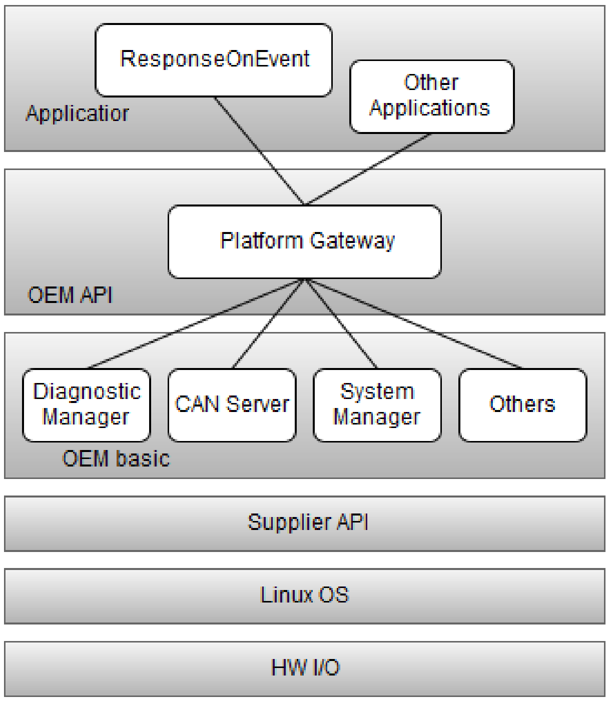

Figure 2.4: Architecture of C300

Figure 2.5: Photo of the C300 used in the thesis

2.3 Diagnostic Trouble Code (DTC)

Each ECU has the ability to diagnose itself and the subsystems it controls and store a diagnostic trouble code (DTC) in its memory if a fault is detected. A DTC can consist of a DTC number, a DTC status byte, a time stamp, a counter and a freeze frame. The freeze frame contains the values of some specified parameters from the ECU from the moment when the freeze frame was stored. For a DTC indicating high engine temperature for instance, the cooling liquid's temperature might be stored in the freeze frame. This data can be used to simplify troubleshooting the system and it can be read out form the ECU using diagnostic services. However, the memory available for freeze frames is limited and an ECU can therefore only store a few freeze frames.

The ECU continuously runs different tests that check for different faults. The test can end with either of the results "Passed" or "Failed" . At a failed test the corresponding DTC is set and in its status byte, the `testFailed` bit is set to '1'. Every time the test fails the counter in the DTC is increased. The DTC's status byte contains seven other bits beyond the `testFailed` bit and they can also be updated due to future tests. They can for instance indicate if the test has failed since the last power on or not. [22]

2.4 Controller Area Network

This section describes the important components of the Controller Area Network (CAN) in short. Since CAN is the communication protocol that was used on the network in this thesis, a description of the protocol is needed. For a more thorough description, see Appendix C.

2.4.1 Protocol and architecture

A Controller Area Network consist of one or more CAN buses that connect two or more ECUs that implement the CAN protocol. Each ECU connected to the same bus will be able to read all the messages on the bus [25]. To address a message to a certain ECU, a field called identifier in the CAN messages can be used. [30] [15]

In-vehicle Controller Area Network often consists of several main buses with the purpose of separating critical components from less critical. Figure 2.6 shows an instance of Scania's CAN network with three main buses. The three buses are joined together by a coordinator unit called COO or Coordinator. [32]

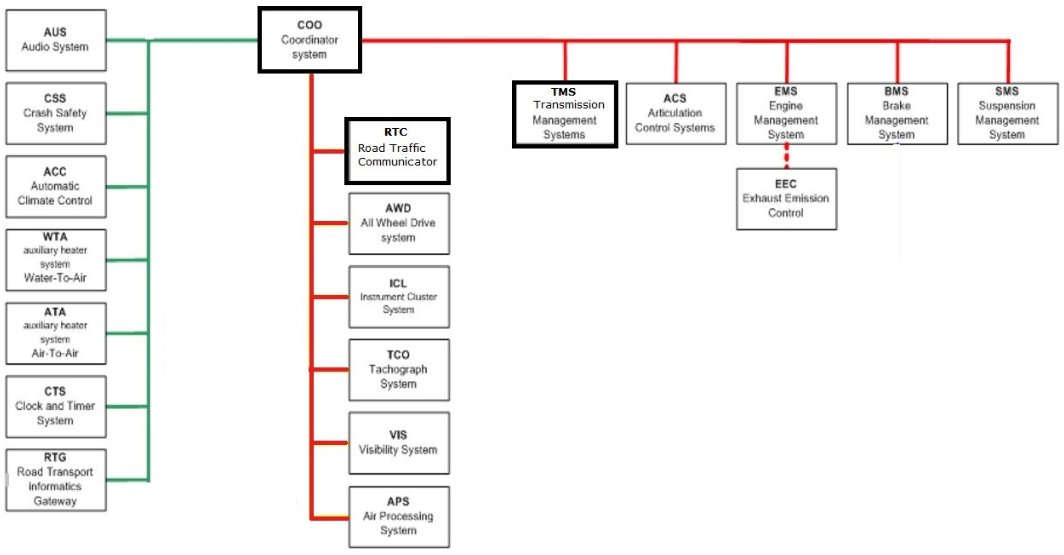

Figure 2.6: CAN network with several ECUs and different buses connected by a coordinator. Here the C300 and the TMS can be seen as well.

CAN allows only one ECU at a time to use the bus for sending messages. Each message that is sent on the bus is called a data frame and it is divided into different fields. One field determines the message's priority and target address and another contains the data such as a diagnostic message or some value like the engine temperature.

2.5 Diagnostic Domains

From the different domains described in section 2.1, three domains were selected to be the main focus in the thesis. They were chosen due to their previous indications of a need for a new type of diagnostic service. They were Remote Diagnostics, Diagnostics of ADAS functions, and Function Development and Verification.

2.5.1 Remote Diagnostics

Remote diagnostics is used to communicate the internal state of the truck to an off-board application using wireless communication. Through the application a fleet manager or a truck owner can look at information for specific trucks he owns. The data that is communicated to the application could for instance be the status of different DTCs. Remote Diagnostics can also be used by help desks to extract information from the truck that can help them troubleshoot the vehicle remotely. A challenge today for Scania's help desk is the fact that the extraction of data from the truck takes too long. It can take between five and fifteen minutes before help desk receives the information from the vehicle and can start working out what is wrong. This is valuable time since it affects the uptime of the vehicle and if the truck is late with its delivery, high fees are charged. Since Scania's help desk has a policy of sending an assistance car to the truck within ten minutes, the information sometimes reaches Scania's help desk too late for it to be of any use. If the assistance car has already departed to relieve the truck when the information reaches the help desk, it will be too late to bring any specific spare parts. If the information of the truck's internal status could been sent to the off-board application as soon as any changes have occurred, this could improve the work of the help desk and valuable time could be saved for the truck driver. [39][37]

One way of improving remote diagnostics would be to have a service that scans the truck for new DTCs and sends them to the off-board application every hour or possibly with an even higher frequency, thus creating a mirror image of the truck. This could improve the work of all services related to remote diagnostics. It is possible today to carry out such a scan but the frequency of the service is limited due to the amount of data traffic that it gives rise to. During a scan for DTCs, a central ECU like the RTC who handles the communication with the off-board application, needs to ask all the other ECUs for their DTCs and their corresponding status. The ECUs must respond by sending all the information concerning their DTCs even if nothing has changed since the last scan. This creates an unnecessary high bus load. To not disturb other more critical applications in the truck, the service must have a low priority. The consequence of a low priority is that the service can take some time to complete each scan which in turn limits the frequency and hence the resolution of the service. [39] [37]

2.5.2 Diagnostics of ADAS Functions

ADAS, short for Advanced Driver Assistance Systems is a collection of advanced functions that are meant to improve the safety while driving. The functions vary between actively engaging in the vehicles behaviour, to alerting the driver of potential hazards. Some examples are automatic emergency breaking (AEB) and warning signals indicating to the driver that he is switching lanes without using the indicator lamps. [40]

As the vehicles are moving towards more autonomous functionality, the ADAS functions gets more dependent on the collaboration between different functions and ECUs to help the system understand what is going on inside the truck or in its surroundings. This makes the functions harder to troubleshoot thus calling for a new type of diagnostics. One way of simplifying the troubleshooting of ADAS functions could be to activate logging of certain data when an ADAS function becomes activated. For example storing the id of the ECU who requested the activation of the brake. Thus if the truck would brake without any obvious reasons, technicians would know where to start looking for the problem. It could also be desirable to log the internal status of certain variables of the ECU or ECUs involved in an activation of an ADAS function. However this would require some amount of memory in the ECUs and the amount of free memory in most ECUs is very limited. [40]

The need for a better way to diagnose ADAS functions is further motivated by the fact that the driver might not always understand how the ADAS functions work. For example a driver could complain about the emergency brake activating without any reason, when it actually is the speed controller that has activated the break to slow down the vehicle. It can be hard for a mechanic to know which functionality that actually requested the activation of the breaks and he might therefore draw wrong conclusions when trying to understand if something is broken in the vehicle. The mechanic might conclude that some parts, like the front radar is broken and needs to be replaced. The lack of good methods to diagnose ADAS functions can thus cause unnecessary costs for the truck owner or a dissatisfaction with the ADAS functions. [40]

2.5.3 Function Development and Verification

To troubleshoot and verify the behaviour of functions under development, logging of data is used integrated into different tools. The logging is done on one of the communication buses in the vehicle and is started for instance by pressing a button or by configuring the logging tool to start when it sees a certain message on the bus. [35]

When logging messages on a bus, it is possible to see which ECU that sent a message and when. The problem today is that in some cases it is not possible to see the reason why the message was sent since this depends on internal ECU data which is not visible on the bus. [33]

The tool used today in this domain at Scania makes use of the CCP-protocol. CCP stand for CAN Calibration Protocol and it is a protocol used for calibrating ECUs. It can also be used to get access to ECU internal data by asking ECUs to send the value of some specified memory address over the bus with a given frequency, thus making the data accessible to the logging tools. This increases the load on the bus and limits the number of ECU-internal variables that can be logged using CCP. The load on the bus also puts an upper bound on the resolution, i.e. with what frequency the data can be transmitted. Furthermore, to set up this type of frequent data transmission one needs to know exactly which build version of the software the ECU is running. This is because CCP uses memory addresses to specify which data the ECU should send and this is likely to differ between different software compilations. Another drawback with CCP is the fact that it cannot be used on vehicles other than test vehicles since the protocol is blocked in commercial vehicles. This is due to the fact that by using the CCP protocol one can access the whole memory space of the ECU and thus overwrite internal variables which is a big safety and security risk. [35][33][34]

2.6 Diagnostic Protocols

A diagnostic protocol defines a set of diagnostic services. By implementing a diagnostic protocol, an ECU can receive, execute and respond to the services in the protocol. An example of a diagnostic service is the readDTCInformation which can be used to read out data connected to DTCs in an ECU implementing the service. If a client wants to extract some information from a server using this service, the client sends a CAN message to the server containing a readDTCInformation request. The server will respond with a CAN message containing the readDTCInformation response. To minimize the message length, the name of the service is translated into a single hex value called a service identifier (SID). According to the UDS protocol, readDTCInformation should be translated to `0x17`.

The two protocols used in the thesis were KWP and UDS. The main differences between the two protocols are the fact that KWP does not define a corresponding event based service like the one found in the UDS standard, called ResponseOnEvent. The two protocols and the general structure of a diagnostic service will be described in this section. [2][22]

2.6.1 The Structure of a Diagnostic Service

The ordinary traffic on CAN consist of non-diagnostic messages. An example is how the engine continuously broadcasts it's temperature so that other ECUs interested in this value can monitor it. The ECU controlling the cooling of the engine for example, is interested in this value. Every time the value reaches the ECU it compares it against a desired value and regulates the cooling of the engine accordingly.

Diagnostic services such as the ones mentioned in 2.1 consist of one request message and one response message. The request is sent by a diagnostic client and is addressed to one or more diagnostic servers. The diagnostic client can be an on-board ECU or a external PC connected to the network by for instance a mechanic. The diagnostic server is an ECU located on-board the vehicle. Upon receiving a diagnostic request the server ECU responds with the corresponding diagnostic response message or an error message. The answer depends on for instance if the format of the request was correct .

2.6.2 Unified Diagnostic Services (UDS)

The unified diagnostic services (UDS) is an international standard for road vehicle diagnostics given by the standard ISO-14229-1 [22]. Most ECU manufacturers are producing ECUs implementing the UDS protocol. All Scania developed ECUs however are using the KWP protocol and therefore this will be the protocol focused on. Even though the protocols are different, it is possible to have ECUs implementing either of them on the same CAN bus.

2.6.3 Key Word Protocol 2000 (KWP)

Scania uses Swedish Implementation standard SSF - 14230-3 [4] which is based on the ISO-standard for the KWP application layer defined in ISO-14230-3 [2]. Since one part of the thesis concerns implementing a prototype of ResponseOnEvent in ECUs using the KWP protocol a brief introduction to the protocol is needed. [2]

On difference between the two protocols is the service identifiers used for different services. In table 2.1 some UDS functions and their counterparts in the KWP protocol are listed. These services are of special interest since they are mentioned in the UDS standard [22] as the only functions recommended to be used in combination with the ResponseOnEvnet service.

A common term used by diagnostic services is CommonIdentifier. This is simply a list of data identifiers which each ECU maps to a certain memory address. A client can thus request a certain data using a data identifier without knowing at which address the data is stored inside the server ECU.

UDS service	KWP service	KWP SID
ReadDataByIdentifier	ReadDataByCommonIdentifier	0x22
ReadDTCInformation	ReadStatusOfDiagnosticTroubleCode	0x17
RoutineControl	StartRoutineByLocalID	0x31
	StopRoutineByLocalID	0x32
	RequestRoutineResultByLocalID	0x33
InoutOutputControlByIdentifier	InputOutputControlByCommonIdentifier	0x2F
ReadMemoryByAddress	ReadMemoryByAddress	0x23

Table 2.1: Recommended services in UDS and corresponding services in KWP

2.7 Diagnostic Domain Analysis Theory

For studies where the research goals are of a qualitative nature, it is generally appropriate to rely on qualitative measures. A description of the qualitative method used to investigate the different domains and the need for an event based service can be seen in appendix D.

2.8 Response On Event

One way to propagate information from one ECU (the diagnostic server) to a diagnostic client (another ECU or a PC) would be to have the client send a request asking the server for the specific information using a diagnostic service. By doing this continuously, the client can monitor values in the server. This process to continuously ask for the same data is called sampling and it causes the same data to be sent several times over the network if the value is unchanged between samples. This can sometimes be redundant and it only causes unnecessary load on the bus. One way to avoid this, is to instead make the server responsible of sending information on the bus at certain events using an event based service.

ResponseOnEvent (ROE) defined in the UDS standard is an event based service and since it is one of the main subjects of the thesis a short summary to it will be given here. For a complete description of the service see ISO-standard 14229-1:2013. [22]

ROE makes it possible for diagnostic clients to define and set up certain trigger conditions and corresponding diagnostic services that should be executed and responded to every time the event defined by the trigger condition occurs. The service to execute and respond to at these times is denoted by serviceToRespondTo and the trigger condition together with its serviceToRespondTo will be denoted by event logic. One example of an event could be the setting of a certain DTC. And a serviceToRespondTo could be ReadStatusOfDTC to transmit that DTC on the bus to the diagnostic client. Figure 2.7 shows how the communication of ROE works compared to sampling.

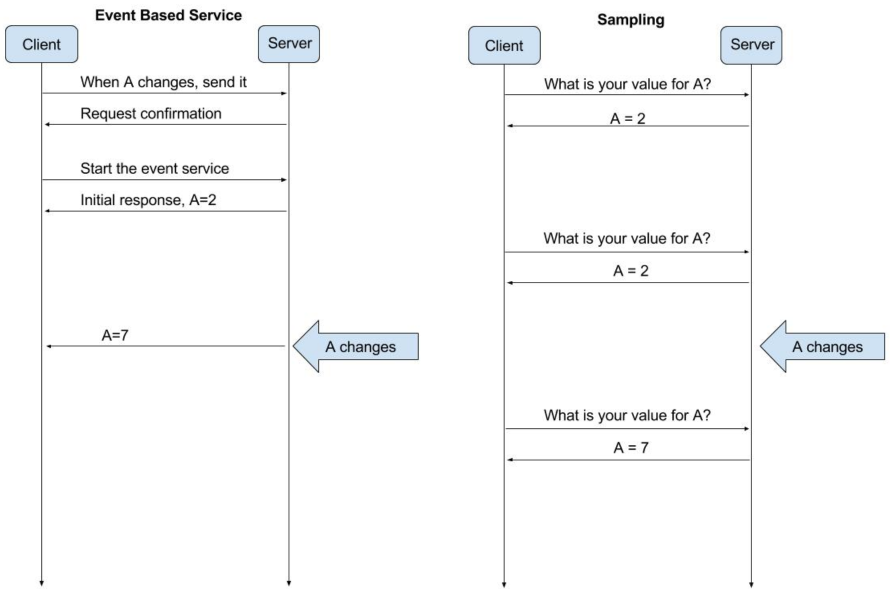

Figure 2.7: ResponseOnEvent compared to sampling

To set up ROE, the client sends a responseOnEvent request message containing the event logic (the trigger condition and the serviceToRespondTo) to the serve. The server answers with a positive or negative response. If a positive response was sent this means that the server has set up the event logic. The service is now initialized. The client can then start the service by sending a start request on which the server responds with a positive response. The service will then be activated and listening for the specified event to occur. Whenever the event does occur the server sends a message to the client corresponding to the serviceToRespondTo specified in the event logic. The server will keep listening for the event and send responses to the client until a certain amount of time defined as eventWindowTime has passed or until the ECU powers down or the client sends a stopResponseOnEvent request to the server. ROE defines a set of possible trigger conditions and serviceToRespondTo. In figure 2.8 a simple overview is given showing the possible trigger conditions and serviceToRespondTo.

The eventWindowTime starts running first when the client has requested to start the service through the startResponseOnEvent request message. When the time defined in eventWindowTime has elapsed the server will send a final response to the client. Such a response is not sent if the service was stopped in any of the other ways. In figure 2.9 the basic behaviour of the service is shown.

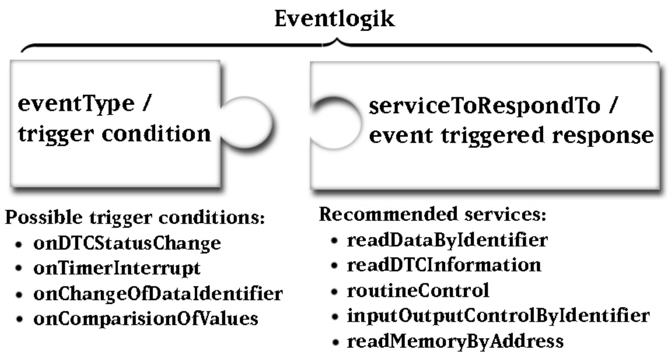

Figure 2.8: The event logic for ResponseOnEvent, with the possible eventTypes and recommended diagnostic services to be used as serviceToRespondTo.

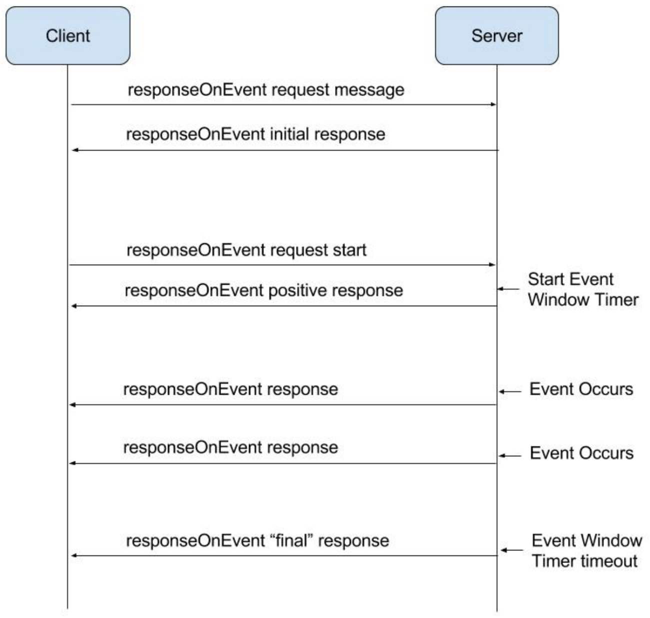
Client
Server
responseOnEvent request message
responseOnEvent initial response
responseOnEvent request start
responseOnEvent positive response
Start Event Window Timer
responseOnEvent response
Event Occurs
responseOnEvent response
Event Occurs
responseOnEvent “final” response
Event Window Timer timeout

Figure 2.9: ResponseOnEvent basic behaviour

The request message for setting up an event logic contains the following parameters:

- eventType: Which event the server should be listening for.
- eventWindowTime: Defines for how long the server shall be listening for the event after the service has been started.
- eventTypeRecord: Contains additional parameters for complementing the eventType.
- serviceToRespondToRecord: Defines which diagnostic service, the so called serviceToRespondTo that should be executed and responded to at the specified event.

The server must evaluate all the parameters in the responseOnEvent request message except for the serviceToRespondToRecord before sending a positive or negative response to the client. If any of the parameters are incorrect the server shall send a negative response. The serviceToRespondToRecord parameter is evaluated first when the specified event occurs. Table 2.2 gives a overview of the request message.

Data Byte	Parameter Name	Byte Value
#1	ResponseOnEvent Request SID	0x86
#2	sub-function = [eventType]	0x00-0xFF
#3	eventWindowTime	0x00-0xFF
	eventTypeRecord[] = [	
#4	eventTypeParameter_1	0x00-0xFF
..	..	..
#4+(m-1)	eventTypeParameter_m]	0x00-0xFF
	serviceToRespondToRecord[] = [	
#n-(r-1)-1	serviceId	0x00-0xFF
#n-(r-1)	serviceParameter_1	0x00-0xFF
..	..	..
#n	serviceParameter_r]	0x00-0xFF

Table 2.2: responseOnEvent request message for setting up event logic.

The parameter sub-function, also called eventType defines which type of event the server shall listen for. This field can also contain, instead of a eventType, a command for starting, stopping or clearing the event logic. Some eventTypes require additional information which is then contained in the eventTypeRecord.

The diagnostic service given by the serivceToRespondTo will be executed when the event occurs. The only diagnostic services that are recommended to be used together with ResponseOnEvent are summarized in figure 2.8.

3 Method

The method used during this thesis can be divided into multiple separate stages. These stages are shown in figure 3.1. Initially a qualitative study was performed in order to elicit the need for an event based service and the requirements on it. These requirements were then used to assess the usefulness of the service ResponseOnEvent defined by the UDS standard. But before that could be done, the standard first needed to be interpreted since it contained some uncertainties. A decision was then made whether or not to look further into ResponseOnEvent or to design an alternative service. A prototype of the selected service was then implemented as a way to investigate the potential problems connected with the realization of an event based service and try to elicit new questions concerning the usability of the service. The chosen service was then evaluated to give a final recommendation of how to design and implement an event based service.

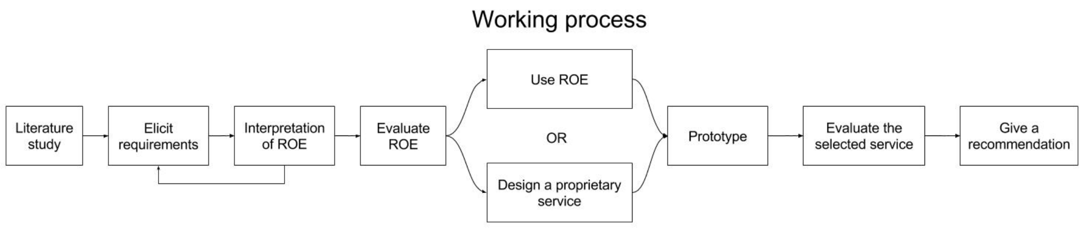

Figure 3.1: The working process applied during the thesis.

3.1 Pre-Study

In order to get a understanding of vehicle diagnostics and the state of the art technology a pre-study was carried out as a first step of the thesis. The pre-study consisted of a literature study and interviews related to diagnostics.

The interviews gave general information about vehicle diagnostics and the technology used in the domain. Keywords for the literature study were also obtained. The literature study was then carried out by searching through a set of chosen databases using the obtained keywords. The chosen databases were: LUBSearch, CRCnetBase, Google Scholar and UniSearch. Some of the keywords were responseOnEvent, vehicle diagnostics, event based service and ECU diagnostics. The found literature was then filtered by reading the abstract, followed by the introduction and results before the whole paper was read. Useful literature was also found through snowballing [24]. The literature study continued throughout the whole thesis as a way to find new relevant information as the research went on. Scania also provided internal documents describing the electrical system of their trucks and standards which were studied in order to understand the technology they use.

3.2 Diagnostic Domain Analysis

In order to find the answer to research question 1, a qualitative research was performed in the initial phase of the thesis to identify how different diagnostic domains could benefit from an event based service and which requirements they would have on an event based service. This section describes how the work was carried out.

3.2.1 Semi-structure interviews

The selection process of subjects to interview was originally emanated from an inventory list of system owners and function developers. To determine which area and experts to interview, guidance from a supervisor at Scania was initially provided as well. The selected subjects were then approached by emails were the theme of the interviews was explained together with some of the questions given in table D.1 so that the interviewee would have time to prepare themselves. This also gave the interviewee the opportunity come up with questions of their own.

Preparations were made by formulating questions before each interview, the questions were adapted to suit the subjects and formulated accordingly to the subjects respective field without being too specific. The questions aimed at elicit knowledge about the present work and problems in the domain as well as to gain understanding for what was realizable using an event based service and where problems might arise. Furthermore the questions aimed to see what was lacking with current diagnostics technique and what requirements it would reflect onto an event based service.

Due to the size of the diagnostic domains it was important to select subjects so that there would be a balance between system owners and function developers from the different diagnostic domains. Due to the size of the diagnostic domains the knowledge and competence are distributed among several employees which meant that some interviews were used to identify areas of interest and were not used as a basis for the requirement elicitation. A total of 23 people were interviewed during the thesis.

3.2.2 Analyzing Qualitative Data from the Interviews

The interviews were always conducted by two persons where one led the interview, while the other took notes and asked additional questions. Conducting the interview in pairs meant that the data could later be verified by comparing the two sets of notations and perceptions. Also the subject generally talks more with two interviewers present which also has an affect on the credibility of the analysis [12].

In some cases when the data of the transcripts were prepared for further analysis additional contact with the subject was necessary to clarify some uncertainties. The subject was generally contacted via email to sort out the ambiguities that were found.

After the interviews, the transcript were studied and the relevant information extracted. From this information different requirements were elicited. The requirements were later presented to the interviewee to confirm that they reflected their needs.

3.3 Interpretation of ROE

In order to understand the limitations of ResponseOnEvent and to be able to assess its usefulness and answer research question 1, the service needed to be studied. The first step was to find and interpret all the uncertainties in the description of the service. This was done by studying the standard and compiling all the uncertainties found and contact the ISO organisation to ask for clarifications. It turned out that the answer from ISO would not arrive during the time of the thesis and therefore the interpretation were done without the help of ISO, using the knowledge gathered so far about vehicle diagnostics. Where possible the interpretation was done to facilitate the fulfillment of the found requirements. As new uncertainties were discovered new requirements needed to be elicit.

3.4 Evaluation of ROE

When the service had been interpreted and understood the evaluation of its usefulness was conducted by looking at which of the expressed requirements that it could support. Implementation details from the service specification was also studied to see if they threatened to clash with the current software architecture and make the service hard to implement. After the evaluation a decision was made whether to follow the standard or diverge from it in order to support additional unfulfilled requirements and fix eventual flaws. During this evaluation the answer to research question 1 was established.

3.5 Prototype Development & Tools

The purpose of the prototype was to examine more in-depth the different ways of how an event based service could be designed and realized and which problems that were connected to the different alternatives in order to answer research question 2 and 3. This section addresses the interested reader who wants a deeper insight into how the development of the prototype was carried out and how it was tested.

The implementation was performed according to an agile methodology in the sense of small increments of functionality and continuous testing and integration between the client and server application. At first, a goal was established describing the key functionality that should be supported. It was then divided into smaller increments for implementation. Overall the implementation was limited to 4-5 weeks.

3.5.1 TMS development

The development of the server application for the TMS was preceded by meetings with representatives from the group responsible for developing software for the TMS. This to get an insight into how the software in the ECU worked and how to set up the development environment and connect the ECU to the PC. Since the software consisted of over 4000 files and many thousand lines of code this contributed to speed up the process of the development process.

The application was developed on a PC and the compiled software was then written to the ECU for testing using a vehicle communication interface (Vci3) which is a device that makes it possible to connect an ECU to a PC. When the software had been written to the ECU and the unit had booted up it was possible to send diagnostic requests to the ECU over CAN using the Vci3 and the program XCOM. XCOM is a Scania developed program and it is enough to say that is contains a window where a diagnostic message can be defined and sent to any ECU on the CAN network connected to the PC through the Vci3.

XCOM had one flaw when it came to developing the event based service and it was the fact that it assumes that the ECU always will respond to each diagnostic request with one, and only one response. If multiple responses would arrive, the first is displayed and the rest is discarded. This made XCOM insufficient for testing the event based service since the event triggered responses would be discarded. To be able to detect all the responses a complementary program was used to log all the CAN traffic which made it possible to detect the extra responses from the TMS.

The development of the client application for the C300 was carried out in a similar fashion compared to the TMS application.

3.5.2 Test environment

Testing was performed continuously during the development. The applications were tested separately and when sufficient functionality was in place they were tested together. To simulate the internal environment of an vehicle the test environment consisted of a C300 located on one CAN bus, a TMS located on another bus and a coordinator that provided the communication link between the two buses. This structure was chosen since it mimicked the real structure in Scania's trucks. One PC was then connected to the TMS's bus through which the TMS was programmed and monitored. Another PC was connected to the C300 through an USB-to-Ethernet connection on the ECU. A photo of the setup can be seen in figure F.3. The server application of the service was also tested in one of Scania's trucks.

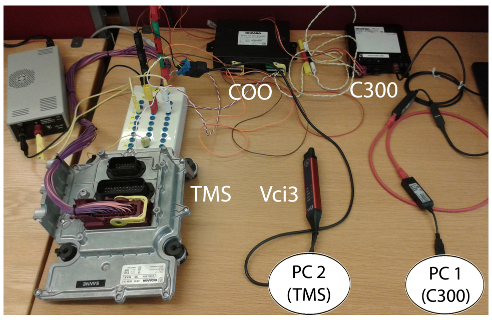

Figure 3.2: The test bench used to develop and test the service. The TMS and C300 were located on two different CAN buses and a coordinator (COO) provided a communication link between the two buses. This structure mimicked the network in a real vehicle.

4 Results

Here all the results found during the thesis will be presented. First a summary of the interviews will be given for each of the different domains followed by the elicited requirements. The interpretation and evaluation of ROE then follows. Then the proprietary service will be described together with the prototype implementation and suggested solutions of how to fulfill some of the requirements which the proprietary service did not fulfill.

4.1 Diagnostic Domain Analysis

An analysis of the collected data from the interviews was necessary in order to identify how an event based service should be designed. It would also give information about advantages and disadvantages of an event based service with respect to the different domains. The results derived from the qualitative research will be presented here. The results are presented for each diagnostic domain separately.

4.1.1 Remote Diagnostics

As described under 2.5.1 the remote diagnostic domain aims to communicate the vehicle's diagnostic status to an off-board application with the purpose of reducing downtime as well as provide services that can reduce the hauliers' costs and increase their efficiency. One found key factor in achieving this is to enable a more continuous update of the mirrored image of the vehicle in the off-board application compared to what is possible today where a complete scan of the vehicle's DTCs are done every 20th hour. To only get a status update every 20th hour leaves help desks unaware of changes for long periods of time and also results in information being lost in between the updates. Freeze frames and time stamps connected to the setting of DTCs might have been overwritten many times in between the scans and only the latest written time stamp and freeze frame will be obtained.

The biggest problem preventing an increased scanning rate is the amount of data that needs to be transmitted on the bus at every scan to decide which data that has changed. If there was a way to keep track inside the ECUs of which data that has changed since the last scan, only new data would have to be communicated on the network, which would contribute to lower the load. But to keep track of what data that has changed would require additional memory. For instance, to be able to only communicate a DTC when a change has occurred in its status byte, a copy of the DTC's old status byte would be needed to compare against. Another alternative is to use a single bit for indicating that the data has been modified since the last scan, but this does not give any information of which bit in the status byte that has been modified. The first alternative is very likely to be too memory consuming to be realizable in today's ECUs due to the large amount of possible DTCs. The later is more likely to be realizable but since the memory is a constrained resource in embedded systems this solution might also require too much memory. These two alternatives were not something that was looked into but the idea highlighted another problem with an event based service. To support the setup and detection of a change in one specific DTC using an event based service would only increase the memory usage of the service by one byte, to save a copy of the DTC's status byte. But to be able to trigger the service on a change in any DTC's status byte would require much more memory since some ECUs have many hundreds of possible DTCs.

One alternative that was seen was to have the event based service receive the DTC's id number every time any function has made a change to a DTC. The event based service could then have a buffer of some fixed size into which the ids were stored. The event service could then send the DTCs in its buffer over CAN and then empty the buffer with some given frequency. Another alternative would be to have each function that modifies a DTC also call the event service and notify it that a change has occurred and that it should check if the change matches any of its trigger conditions. Both these alternatives would require all the functions that can modify any DTC to also call an additional function related to the event based service. This could increase the execution time of DTC-modifying services and the scheduling of services in the ECU might have to be modified as a consequence.

4.1.2 Diagnostics of ADAS-functions

The need for an event based service for diagnostics of ADAS-functions was motivated by the need of a new way to diagnose functions with an increasing complexity. One of the advantages of using an event based service is just as for Remote Diagnostics, the possibility of decreasing the network load on CAN during the diagnosis. Another benefit is the flexibility that an event based service could provide. It was clear from the domain experts that it always has been hard to predict which problems that might occur and which methods that would be the most efficient for troubleshooting. Therefore an event based service could prove to be of great value due to its flexibility and versatility.

At the activation of certain ADAS-functions in an ECU, the ECU can saves some data connected to the function. For instance when the AEB is activated, a picture of the area in front of the truck is saved in the ECU controlling the AEB. Since some ADAS-functions are located in one ECU but the activation request comes from another ECU, the values that caused the activation of the function are not always accessible and can therefore not be saved. But if these values could be communicated to an ECU responsible for the ADAS-function, troubleshooting could be facilitated. This type of data communication could also make it possible to store relevant data inside a completely different device that has nothing to do with the activation of the ADAS-function. This could be useful if the ECU responsible for the ADAS-function does not have enough free memory to save the data of interest. The data could for instance be stored in the C300 and later sent to an off-board application. In such a way, it would be possible to build up a database of different incidents where ADAS-functions had been activated and what has caused the activation.

4.1.3 Function Development and Verification

There is room for improvement in today's logging tools according to the experts interviewed and an event based service has some beneficial functionality that makes it interesting for the domain. To be able to use an event based service without having to take the compilation version of the ECU's software into account as well as the potential of being able to use the service in commercial vehicles gives the event based service an edge compared to CCP. The service also presents the possibility of decreasing the network load compared to CCP which makes it possible to log more ECU internal data. The fact that the data would be communicated per event also makes a higher resolution possible compared to if the data would be sent periodically.

Some of the limitations when it comes to using an event based service for logging purposes is the fact that internal logging inside the ECU is still limited by the amount of free memory in the ECU. Therefore the best way to log the data using an event based service is still to send it on the CAN bus and use a logging tool that logs the CAN traffic to pick up and store the data. Furthermore the event based service is not integrated in the tools used today and therefore would have to be set up using other tools which might cause some additional work for the tester. But the experience gained from the prototype showed that the service could be set up using scripts in on of Scania's software tools which could be used to limit the effort.

4.2 Elicited Requirements

Here the elicited requirements are presented with a short context so that the reader can more easily understand their origin. The requirements have also been given a reference to the domain from which they were derived. The references and corresponding domains are Remote Diagnostics (RemD), Function Development and Verification (Dev) and Diagnostic of ADAS-functions (ADAS). Some of the requirements were later excluded due to the fact that they did not directly concern the event based service, but the tools that will use it.

4.2.1 Setup, Start and Stop

-If it is possible to store events in an ECU between power on and power off, vehicles could be sold with default event logic already set up. But to be able to activate just this logic and not any other event logic in an ECU it must be possible to start a single event logic. It would also be beneficial to be able to clear specific event logic.

R 1 `Start, Stop and Clear a Single Event Logic:` It must be possible to start, stop and a clear single event logic in ECUs. (RemD)

-If as much as possible of the service can be controlled remotely, the uptime of the vehicles could be increased compared to if the vehicle would have to visit a workshop to manipulate the service.

R 2 `Remote Start and Stop:` It must be possible to remotely start and stop specific event logic in ECUs. (RemD) (Dev)

R 3 `Remote Setup:` It must be possible to remotely setup new event logic for the service. (RemD)

-If the ECU is processing an event and an identical event occurs triggering the service, the second event should not be discarded since it can be of interest.

R 4 `Multiple Events:` If the ECU is processing one event and an identical event occurs in the same ECU, the service must process both. (RemD) (Dev)

-To simplify the setup of event services in multiple vehicles it would be preferable if whole configurations of the service could be saved in a PC environment.

R 5 `Vehicle Configuration:` It must be possible to save whole vehicle configurations of the event service on a PC. (Dev)

-To facilitate the synchronization of different CAN-logs from different vehicles it would be good if event responses could be time stamped.

R 6 `Time stamps:` It must be possible time stamp event responses. (Dev)

4.2.2 Storing event logic

-It would be good to have predefined event services that are stored in the vehicle's ECUs ready to be activated at a certain occasion. Event logic could be set up during the production of the vehicle or flashed down to the vehicle during a workshop visit or remotely.

R 7 `Store Logic:` It must be possible to store event logic in ECUs between power off and power on. (Dev)

4.2.3 Event Triggers

-The service should be able to react to the setting of DTCs to help to communicate DTCs to an off-board application as soon as possible.

R 8 `Trigger on DTC:` It must be possible to use DTCs as event triggers. (RemD) (ADAS)

-By using boolean combinations of ECU data as trigger conditions, more complex trigger conditions could be created making the service more dynamic. For example could the triggering of events during certain conditions be prevented such as when the engine speed is equal to zero. This is useful when certain spare parts are changed during workshop visits since this might cause error codes to be set when there actually is no fault.

R 9 `Comparison Logic:` It must be possible to compare multiple data using any of the following comparison logic: "bigger than", "smaller than", "equal to" and "not equal to" as event logic. (RemD) (ADAS) (Dev)

-It would be good if event trigger conditions could be updated when an event occurs such as first trigger on "value > 1" and in case the condition becomes fulfilled, change the condition to "value > 2". This would make the service more dynamic.

R 10 `Update Trigger Condition:` It must be possible to automatically update a trigger condition after a trigger has occurred. (Dev)

-To be able to trigger on more complex errors, a combination of comparisons as trigger condition would be good such as "if (value1 == 2 AND value2 == 6)".

R 11 `AND/OR:` It must be possible to combine different comparisons using the following logical operations: "AND", "OR". (Dev)

-Data interesting to be used as trigger conditions is accessible through the diagnostic function ReadDataByCommonIdentifier.

R 12 `ReadDataByCommonIdentifier:` Is must be possible to trigger events on ECU internal data accessible through the diagnostic function ReadDataByCommonIdentifier. (RemD) (Dev)

-Experts working with function developing and verification expressed a desire to be able to access ECU internal data through the service without having to know at which address that data is saved.

R 13 `Address Independent:` One must be able to set up the service to access ECU internal data without having to know at which address that data is stored. (Dev)

-If events could be triggered by contradicting messages on the CAN bus like brake and acceleration from different sources like the automatic emergency brake and the driver, it would be possible to provide information about when a driver tries to override an ADAS function.

R 14 `CAN-Frames:` It must be possible to trigger events on certain CAN frames. (ADAS) (Dev)

4.2.4 Sessions

-To be useful to Remote Diagnostic, the service needs to work on commercial vehicles while they are in service.

R 15 `Default Session:` The service needs to work at least in the default session. (RemD)

-ADAS perform their diagnostics in the extended session.

R 16 `Extended Session:` The service needs to work in the extended session. (ADAS)

4.2.5 Logging of Data

-To facilitate the logging of ECU internal data using the current logging tools available, the service should be able to send ECU internal data like DTCs on CAN at certain events.

R 17 `Internal Data on CAN:` The service must make internal ECU data available on CAN. (Dev)

-To facilitate troubleshooting and verification of functions, it should be possible to store ECU internal values before and after an event. For example to store the temperature of the cooling liquid before and after the DTC for high engine temperature is set.

R 18 `Logging:` The service must provide functions for saving the values of internal variables before and after an event. (Dev) (ADAS)

4.2.6 Performance

-It is desirable that diagnostic services that should be used during uptime of the vehicle should take up as little as possible of the CPU power in the ECUs to not disturb other functionality. Representatives for the TMS said that the increased CPU-load must not exceed 1%.

R 19 `CPU Usage:` The increased CPU usage in the TMS must not exceed 1%.

-The TMS has about 1.6MB free memory. Since the service shall be implemented in the TMS without any new hardware support, the service must not exceed that number. Other ECUs have much less free memory so it is desirable if the service could be kept as small as possible.

R 20 `Memory Usage:` The amount of memory that the service takes up must not exceed 1.6MB since that is what is left in the TMS.

-To facilitate the detection of secondary faults, it must be possible to conclude in which order events have occurred. If for instance multiple DTCs are set, it would be good to know which

R 21 `Order of Events:` It must be possible to derive in which order different events from different ECUs occurred using the service. (Dev) (RemD)

4.2.7 Multiservers

-For the service to be useful to Remote Diagnostics the service needs to provide a way to collect DTCs from multiple servers and send them to an off-board application.

R 22 `Multiservers:` The client must be able to set up event logic in multiple servers and listen for responses from them at the same time. (RemD)

4.2.8 Multiclients

-If a service would be stopped due to a session change in the server ECU and not started again when the ECU returns to the default session, the service might miss to report events.

R 23 `Stop on Session Change:` If a service is set up and started in session X, a session change in the server must not cause the service to stop and not start again when the server returns to session X. (RemD)

4.3 Interpretation of ROE

The uncertainties found concerning ResonseOnEvent in the UDS standard are summarized in appendix A. Since the answer from ISO did not arrive in time for this report the uncertainties were interpreted using the gathered knowledge during the thesis.

4.4 Evaluation of ROE

The comparison of ResponseOnEvent with the gathered requirements was done in order to find the answer to research question 1. The comparison is summarized in table 4.1. There were some requirements that the service did not fulfill and some which the service did not specify anything about. One could possibly make extensions to the service to support those requirements which the standard did not specify anything about and still claim to follow the standard. But the requirements which clearly differed from what was stated in the standard would have to be left out. For example one could add the functionality to start and stop single event logic since there were reserved sub-function identifiers intended for the vehicle manufacturer to use. But to make changes to the message format of the service would not be possible. Therefore adding time stamps to the event responses would not be possible if the standard should be followed.

The most important requirements not supported by ResponseOnEvent were the ones concerning more advanced trigger conditions (R 11, R 14), the requirements concerning logging inside the ECU (R 18) and the requirement concerning time stamps (R 6). These requirements were considered to be the most important non-fulfilled requirements since they concerned functionality that had been central in the discussions with the different domains. Therefore an attempt to fulfill them was considered valuable. It is important to state that beyond fulfilling the requirements there was another reason to consider an alternative to ResponseOnEvent. That reason was to take height for possible future needs, by for example adding support for additional diagnostic services to be used as serviceToRespondTo such as the diagnostic service ReadMemoryByAddress. This was considered a flexible and powerful diagnostic service to have access to through the event based service.

In conclusion there was enough room for improvements on ResponseOnEvent to make it worth trying to design an alternative service. The answer to research question 1 was therefore that ResponseOnEvent could not fulfill all the requirements and an alternative service could be of interest. To answer research question 2 , an alternative service was therefor investigated. The alternative service, also called the proprietary service was made to follow ResponseOnEvent as closely as possible but they diverge at some points. Exactly what differences there are will be pointed out in section 4.5. The reason why the proprietary service was chosen to resemble ResponseOnEvent was since a lot of time could be saved by only making changes to an already existing description of an event based service instead of designing a completely new one. Therefore the proprietary service could be called an extension of ResponseOnEvent but with some limitations and changes in the design.

Support	No support	Not specified
	R 1	
		R 2
		R 3
R 4		
		R 5
	R 6	
R 7		
R 8		
R 9		
	R 10	
	R 11	
R 12		
R 13		
	R 14	
R 15		
R 16		
R 17		
	R 18	
		R 19
		R 20
	R 21	
R 22		
R 23		

Table 4.1: The expressed requirements compared with ResponseOnEvent.

4.5 Proprietary Service

Here the answers to research question 2 will be presented in the form of a description of choices regarding the design of the proprietary service together with the reasons behind them and their consequences. The complete proprietary service can be seen in appendix B. Since a lot of the design choices were made in order to solve different problems the answer to research question 3 will also partially be given here. Some of the problems that were discovered during the implementation of the prototype will also be described in chapter 4.6.

4.5.1 Requirements

To decide which requirements to focus on when designing the proprietary service, the requirements were sorted into different groups. One group contained all the requirements which ResponseOnEvent already fulfilled. It was easy to verify that the proprietary service supported these as well since it was an extension of ResponseOnEvent. Another group contained all the requirements which were deemed as out of scope of the service or to complex to design a good support for during the limited time of the thesis. The last group contained the requirements which were deemed as realizable and not to hard to include in the design of the proprietary service.

- **Already Supported**: R4, R7, R8, R9, R12, R13, R15, R16, R17, R22, R23.
- **Out of Scope/Too complex**: R2, R3, R5, R10, R11, R14, R18, R21.
- **Included in the proprietary service**: R1, R6, R19, R20.

To start, stop and clear single event logic (R1) was simple to include in the design of the proprietary service since ResponseOnEvent left some eventTypes to the vehicle manufacturer to define which could be used to fulfill the requirement. To add time stamps (R6) required a reconstruction of the event responses and thus not compatible with ResponseOnEvent. The requirements concerning the memory usage and the CPU-load (R19, R20) needed to be supported for the service to be realizable in the current architecture and therefore a must-include.

The requirements concerning remote start, stop and clear of the service (R2, R3) was something that was considered out of scope of the service since it had nothing to do with how the service itself should work. To be able to save a vehicle's configuration of the service on a PC (R5) was also considered out of scope since it would not affect the structure of the service.

To update trigger conditions upon an event (R10) and to combine logical comparisons (R11) would not be hard to support but to do it in a good way that minimizes the message size and facilitated some amount of flexibility would require a more thorough knowledge about the intended use cases. Therefore only a short recommendation about how to support this requirement was given, see section 4.7.

An ECU does not take in and process all the messages on the CAN bus since this would cause unnecessary CPU-load. Instead a filter is used in the hardware to filter out the messages that are of interest to the ECU. To be able to trigger events on certain CAN frames (R 14) the filter would in some cases have to be modified to let more messages through to the CPU. The possibilities to do this and its consequences were unclear and therefore this requirement was left out from the design of the proprietary service.

The logging of data inside ECUs, before and after an event (R 18) was excluded from the design of the proprietary service since there existed no diagnostic service which could be used as serviceToRespondTo to communicate the logged data to the client at an event. In 4.7 possible solutions for both R 18 and R 14 are discussed in more details.

The requirement concerning the order of multiple events from different ECUs (R 21) was left out since it seemed impossible to fulfill in a sufficient way since it would require a global ordering of all the events in the system. The fact that time stamps of event responses was made possible made this requirement less critical.

4.5.2 Design Choices and Limitations

The proprietary service strives to follow ResponseOnEvent as closely as possible but some changes were made to make it fulfill four additional requirements. The aim was also to make the service more flexible so that it hopefully would be able to meet future needs. In this section the most important differences between the two services will be pointed out together with their consequences.

Since one of the biggest obstacle with the realisation of an event based service turned out to be the limited amount of free memory and processing power in ECUs, the design of the proprietary service was a balance between making the service as flexible as possible while still trying to keep the complexity of the service to a minimum.

One trigger condition per serviceToRespondTo

During the interpretation of ResponseOnEvent it was concluded that is was only possible to set up one trigger condition per serviceToRespondTo. If it would be possible to set up multiple trigger conditions for each serviceToRespondTo this could lead to instability problems. If two client applications in a client ECU sets up two different trigger conditions using the same serviceToRespondTo, the client ECU will not be able to tell which condition that has been triggered when it receives the event generated answer. Figure 4.1 describes the problem with the trigger conditions "A > 4" and "B == 0". The serviceToRespondTo is simply denoted by X. When receiving X the client ECU can not know which application waiting for an event triggered message of the type X that it should notify, since both of the applications are waiting for X but for different reasons.

The proprietary service follows ResponseOnEvent and does only allow one trigger condition per serviceToRespondTo. When a client requests to set up a new trigger condition for a given serviceToRespondTo, the old trigger condition will be overwritten. Thus when the response X arrives to the client ECU, it can be sure of which event trigger condition that has been fulfilled.

Another alternative for updating the trigger condition would be to reject the new trigger condition until the old one is stopped or cleared. The only motivation for not using this design was to make the service easier to set up and change.

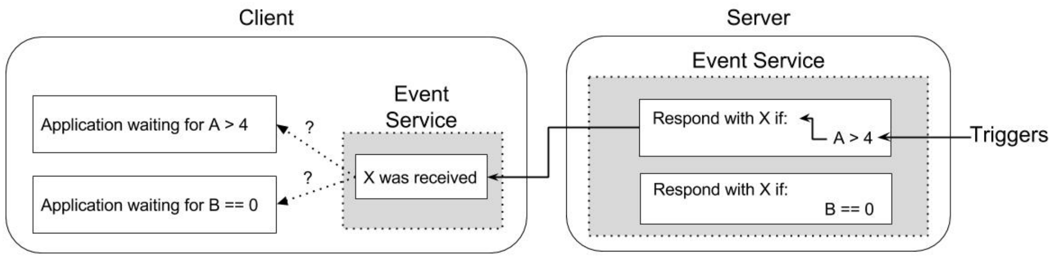

Figure 4.1: What would happen if multiple trigger conditions (A > 4 and B == 0) would be combined with the same serviceToRespondTo (Send X).

The decision to only allow one trigger condition per service to respond to simplified the design of the service and prevents potential problem. However, it also makes the service less flexible. One alternative would be to not limit the number of times the same serviceToRespondTo is combined with different trigger conditions and leave it to the client to solve any potential problems when uncertainties arises concerning which condition that was triggered.

This could increase the flexibility of the service but it would also increase the complexity of it. It would for instance require some type of indexing of the event logic in the server to facilitate the starting and stopping of specific event logic since they no longer would be uniquely identified by their serviceToRespondTo.

Multiclient

In ResponseOnEvent nothing was mentioned about the service's behavior when there were multiple clients on the same network using the service. ROE was possibly only intended to be used by one client at a time. But it was not hard to imagine a future where the different domains looked into during the thesis could require the use of the service at the same time. Remote Diagnostics might want to mirror the vehicles internal state using the service and at the same time the service could be needed by Diagnostic of ADAS-functions in order to store data connected to the activation of ADAS-functions. Due to this possible need the service's behaviour in cases of multiple clients was included in the proprietary service.

The support for multiple clients introduced new complexity to the service. One example is if a client sets up an event logic in one server and then another client tries to start that logic by sending a start request to the server. Then who would get the responses to the serviceToRespondTo? It was decided that it should be the first ECU who set up the logic that would also get the responses. One could also simply prevent the different clients from interacting with eachothers event logic but no reason was seen to prevent this type of behaviour. So to not limit the service's flexibility, the proprietary service allows clients to start, stop and clear eachothers event logic. This prevents event logic from being permanently set up in a server if the client is removed from the network without clearing its event logic. Some logic needs to be implemented in the client so that this type of interaction between clients does not cause any hazard. There is for instance the risk of two clients continuously overwriting eachothers event logic. To prevent this, the person configuring the ECUs must have a good knowledge about which ECU utilizes which serviceToRespondTo.

When the number of event logic setup in servers increases, it will become important to keep track of each truck's individual configuration so that different applications are not designed assuming the possibility to use a certain serviceToRespondTo in an ECU which another application already is using. This is not something that will be looked into in this thesis but a found issue that might need to be addressed if the service starts to be used for more long term solutions and not simply for setting up a test during a short test run.

Time Stamped Responses

To be able to establish a global order of event as specified according to the requirement R 21 proved to be difficult to fulfill. The difficulty lies in the fact that the diagnosis message has a lower priority and can therefore be delayed out on the CAN bus if a controller is busy with higher priority tasks or if the bus is occupied by other messages. Instead of arranging all the events in the system in a global order, the proprietary service introduces the possibility to have the server time stamp the serviceToRespondTo responses. This will make it possible to get information concerning when the event occurred even if its response message becomes delayed out on the bus.

Since some ECUs lack an internal clock it limits the accuracy of the time stamp down to the synchronization messages available on CAN. These are messages that are sent with one second intervals to help the ECUs stay synchronized. The resolution in time for most ECUs is therefore limited to these messages. But if more accurate internal clocks would become available this extension could become very valuable both for Diagnostics of ADAS functions and logging purposes since it could be possible to see in which order different event triggered and compare logs from different vehicles. Even if high accuracy clocks are added to each individual ECU, there is still an upper bound for the resolution due to the fact that each ECU needs to synchronize its clock towards a global clock and this process will always introduce some minor error due to variations in the delays of the communication.

The time stamps will be added to the end of the diagnostic messages thus changing the format of the message as they were defined by ResponseOnEvent and the KWP protocol. To deal with this new message format might require some additional work when implementing the service.

ReadMemoryByAddress

The diagnostic service ReadMemoryByAddress is a powerful service since it can be used to access memory areas of the ECU which have not been given an identifier. ReadMemoryByAddress is however not mentioned in ResponseOnEvent among the recommended services to be used together with the event service. The reason might be the fact that the service takes a parameter denoted `MemorySize`, which defined how much data that should be read. If the amount of data to be read it too big, the execution of the service might take to much time to execute which will impact the whole ECU's performance. Thus not all diagnostic services are fit to be used together with an event based service due to the real time constraints in an embedded system. But since ReadMemoryByAddress was such a powerful service it was included in the proprietary service and a note was left in the proprietary service that one must taken the execution time of the serviceToRespondTo into account. If the execution time takes too long there is a risk that other applications running in the ECU will be affected and the ECU might crash.

Some addresses in ECUs can only be read using the service in combination with security access. But according to an expert at Scania [43] most addresses can be read without security access. To read addresses protected by security access could be problematic since the password might not be located inside the vehicle. If ReadMemoryByAddress would be setup to read an address protected by security access, an error message would be sent instead of the desired response every time the event logic is triggered, if a security access has not been performed. At these error messages one could possibly perform a security access to read the protected data. To keep the server ECU in a security accessed state while waiting for the event based service to trigger is another alternative but it could be problematic since the security access might require the vehicle to be standing still for instance, which might limit the service usability. To keep the sever in a security accessed state also opens up the possibility for other clients to access protected services which could prove a threat against the security.

4.6 Prototype

The purpose of the prototype was to investigate various implementation alternatives and to build knowledge about potential problems related to the implementation, thus answer research question 3. The prototype could also help to elicit questions for further investigation.

4.6.1 C300

Acting as a diagnostic client the main purpose of the C300 are to send and receive diagnostic messages. Functionality that supports sending and receiving diagnostic messages was already implemented which was why the prototype focused on trying to utilize the existing functionality. The problem of not being able to receive multiple responses, as explained under section 2.2.3, meant that restructuring the diagnostic manager was necessary in order to support functionality for an event based service. A system description of the prototype can bee seen under appendix F as well as a sequence diagram explaining the process of sending and receiving messages.

The prototype application developed for the C300 supported the actions to send a request message to the TMS, requesting to set up a given event logic and to start this using a second message. The C300 application could then send the incoming event triggered responses to the PC connected to it, which made is possible to observe the messages in a terminal window.

One problem with the existing software architecture was that it assumed that each diagnostic message would generate only one response from the server. When an application in the C300 wants to transmit a request message on the CAN bus it calls the Diagnostic Manager which transmits the message and waits for the response to arrive. When the response arrives it is mapped to the application which sent the request. Only one application at a time can await an response like this, making the mapping simple. When the response has been handled the Diagnostic Manager is again free for other applications to use. The Diagnostic Manager discards each incoming request which can not be mapped to a pending request. This way each event triggered response would be discarded unless the Diagnostic Manager is given an additional requests to which the event responses could be mapped. But by doing this the Diagnostic Manager would be blocked and no other applications would be able to send any messages until a response message has arrived which could be mapped to the pending request.

The prototype was designed to circumvent this by notify the application that implements the proprietary service when any incoming response were about to be discarded. Thus each unmapped response message caused a notification of the waiting application which is something that needs to be dealt with to not cause unnecessary notifications and CPU-load. One way of doing this could be to check if the response message matched any of the services that the event based service could generate, and only in those cases notify the application. To make the event based service accessible to other applications in the C300 an API could be designed through which they could request to set up event logic and receive the responses.

4.6.2 TMS

Here a description of the server prototype application will be given. The most important implementation choices will also be highlighted together with their consequences.

Supported Services

There was no time to implement support for all the requirements in the TMS. Therefore only some of the key functionality were implemented. The finished prototype application supported the following tasks.

- Set up event logic with the eventType: onChangeOfDataIdentifier and the serviceToRespondTo: readDataByComonIdentifier.
- Start, stop and clear the event logic.
- Send the initial response to the setup request message.
- Detect a fulfilled trigger condition and cause the execution of the serviceToRespondTo.
- Respond to the request reportAllActivatedEvents.
- Store the event logic between power on and off using the ECU's non-volatile memory.

Implementation Details

Since diagnostic services always consist of a request and a corresponding response, there had been no need when designing the current software architecture to support the transmission of unrequested responses from the TMS. Therefore it was not possible to send event triggered responses using the current architecture for diagnostic services. To make this possible would require a restructuring of the software architecture, which according to a software architect at Scania could take up to a month only to get the service running [44]. The amount of time required for implementing regression tests and to verify the service could also take a considerable amount of time. There were two other alternatives which both were not considered alternatives for a final solution since they went against the logic in the software architecture. The first alternative was to use something called the "general purpose CAN API" which made it possible to send an arbitrary CAN message on the bus. The other alternative included calling a function for adding data to the outgoing message buffer. To be able to continue with the prototype implementation the general purpose API was used since it was considered easier to get started with compared to the other alternative. To give a overview of the final software design of the prototype, a couple of sequence diagrams showing the execution of the prototype can be seen in appendix E. [44]

The checking of the trigger condition was done in a loop running every 10ms. This specific frequency was chosen because there already existed such a loop in the architecture. It was possible to create new such loops with different frequencies but it required a good understanding for the software architecture. The frequency of the loop determined the worst case delay between the fulfillment of a trigger condition and the triggering of the event and is thus what puts an upper limit on the resolution of the service's time stamps. With a loop that runs every 10ms the worst case delay would be 10ms. To get a higher resolution one could choose a higher frequency but this also increases the load on the CPU which is a highly valuable resource in almost any embedded system.

Among the requirements there was a requirement stating that the service should not cause more than a 1% increase of the CPU-load, (R 19). The CPU-load that the prototype caused was estimated to 0.15%. This was estimated by measuring the time it took the prototype to detect an event and execute the serviceToRespondTo. The time was then divided by the loop time for the service which was 10ms. This gave a value of how much of the 10ms that the CPU spent executing the service, which is the same as the CPU-load. Using the prototype's CPU-load as reference it was deemed possible for the full implementation to use less than 1% of the CPU. Depending on the resolution required of the service and the CPU-load it is allowed to cause, the frequency of the loop can be changed.

Instead of using an if statement running in a loop checking for new fulfilled trigger conditions every tenth millisecond, it could be possible to add a function pointer to the relevant functions in the TMS making it possible for these functions to trigger the execution of the event based service. An example is to let the function handling the writing of new values to variables in the ECU also start the execution of the event service which then checks for new fulfilled trigger conditions. This would increase the time resolution of the service but it could also cause an increase of the CPU-load in the same way as an increased loop frequency would. Using this solution could contribute to lower the power consumption of the ECU if it would be possible to let the ECU move into some type of sleep mode between the execution of other services when no loop is running every tenth millisecond checking for new fulfilled trigger conditions.

Another constrained resource in embedded systems besides the CPU-load and the power consumption is the memory. The amount of flash memory that the prototype consumed was about 3kilobytes. This was the difference in amount of used flash memory that the building script calculated with and without the event based service. To implement the complete proprietary service was estimated to take about four times that memory since the prototype had partly support for two of the seven different suggested serviceToRespondTo. The total amount of memory that the service would need was thus estimated to 12 kilobytes. This was a very rough estimate but since this was far below the limit of 1.6MB (R 20) it was deemed that it would be possible to implement the complete service using the flash memory current available in the TMS.

4.7 Suggested Solutions for Other Requirements

Here some potential solutions are listed of how to support the requirements that were not considered in the proprietary service.

- R 10, R 11: These requirements could be supported by extending the setup request message for the eventTypes onComparisionOfValue with more fields. Nothing shows on any complication in doing this but it might require some extra research so that it is done in a way that minimizes the message length while still offering enough flexibility. It is also important that the checking of the event does not take too long so that the service has time to finish its execution before its next loop iteration, otherwise the ECU will crash.
- R 14: To make it possible to trigger events on certain CAN frames might require a modification of the ECUs message filter if the event should be triggered by a message that the ECU usually does not read. This could potentially increase the CPU-load since the ECU then would read more messages. It is also not clear if there could be any problems with ECUs starting to listen to new messages than what they usually do [42]. This is a requirement that seems tough to fulfill without a more thorough study focused in this specific area.
- R 18: No difficulties could be seen in fulfilling this requirement, but just as for R 10 and R 11 a more thorough insight into the specific use case of the service would be needed to make the service as flexible and memory efficient as possible. Since the amount of free memory is often small, it is likely that this service would be deemed too memory consuming to be implemented if not the value of the service could be clearly proven.

  To send the logged data to the client at an event, one could possibly use one of the existing diagnostic services that are used for reading data, such as ReadMemoryByAddress. Even if the client have no idea about at which addresses the logged data ends up in the server, the server could still use this service simply as a way of communicating the data as long as the client receiving the data understands how to interpret it. To give the logged data a common identifier would be another alternative but the amount of common identifiers is limited and thus should be used with care.
- R 21: The requirement concerning the order of multiple events from different ECUs seemed impossible to fulfill in a good way since it would require a global order of events in a system where the events could occur inside an ECU and be sent on the bus first after some time has passed. The requirement was therefore deemed as to hard to fulfill.

4.8 Applicability to Other ECUs

The prototype had only the TMS and C300 as target ECUs but by taking other ECUs into consideration, the design for the service could be done in a way that facilitated the usage of the service in other ECUs. One difference that was investigated was the two different software used at Scania for supplying the applications with platform services. But the expert talked to meant that the two different software both were able to support the same services since they both used the same underlying platform [41]. In conclusion, no problems were found that would make the realization of an event based service harder in other ECUs compared to the ones used in the thesis with respect to the different software running in them.

It is however uncertain how well the service would work on other ECUs due to the amount of memory and processing power that it requires. Since most ECUs are designed and optimized for a specific purpose, it is often very little free memory and CPU-power left in them since this would be a waste of resources. Some ECUs only have a few bytes left while others might have a bit more. But even if the amount of memory left in the ECU would be enough for an event based service, it is very unlikely that those remaining bytes would be used for implementing a diagnostic service instead of some other more important functionality. [46]

5 Discussion

In this section the method for carrying out the study will be critically discussed with the purpose of highlight shortcomings that might have affected the outcome. The pros and cons of introducing new diagnostic messages to support an event based service will also be discussed as well as the fact that the service introduces non-deterministic network load which can be a problem if bad trigger conditions are set up. Lastly some areas for future research will be discussed.

5.1 Method

The interviews were one of the main sources of information during the thesis since very little documentation was found of the software system architecture and the work in the different domains. When relying on interviews there is always the risk of misinterpretation and that not all the facts of interest are brought to light. During the interviews it was easy to lose focus of the actual intent of an event based service as the interviewee started to come up with possible ways of using the service without having completely understood its limitations and how it was meant to work. A lot of different use cases for the event based service were suggested which later proved to be easier and more preferable to implemented using other services. Due to this, some additional interviews were required to finally be able to show that the domain would benefit from an event based service such as ROE.

The aim of the thesis was to examine if the chosen domains could benefit from an event based service by comparing their needs to ROE. Since ROE only represents one way of designing an event based service this limited the way an event based service was thought of which might have affected the requirements elicitation. By only looking at the aspects related to the functionality present in ROE other important requirements might have been missed. But to limit the scope of the thesis some assumptions and limitations was needed regarding what an event based service was and what should be investigated. The proprietary service was based on ResponseOnEvent and it is therefore possible that better solutions could have been found if a completely new service had been made from scratch or if other event services had been investigated besides ROE. But since no other alternative designs for event services were found during the literature study, ROE was the only alternative to investigate.

When starting out the requirements elicitation a complete understanding of the service and the domains was still not achieved causing some of the initial requirements to later be deemed as out of scope when a better understanding for the service's limitations and intentional usage had been acquired. These requirements were not removed from the thesis as they could prove to be of value for future research. The initial interviews were carried out with the purpose to both gain knowledge of the domains as well as to elicit requirements since this was thought of as more time efficient. If a better understanding for the domains had been achieved before the requirements elicitation it is possible that requirements of higher quality would have been acquired.

The implementation of the prototype showed on many different problematics both in the design of the service and the surrounding software architecture. Since the software architecture differs between vehicle manufacturers it is likely that other problems would have been found if the implementation had been done in another system at another vehicle manufacturer. It is also likely that depending on which functionality that was focused on in the prototype implementation, different limitations and problems would have been discovered.

5.2 ROE and the Proprietary Service

Here the found answers to the research questions will be discussed.

How does ResponseOnEvent defined by the UDS-standard meet existing needs of diagnostics on heavy duty vehicles?

ResponseOnEvent fulfilled some of the requirements that were found in the different domains and could prove to be a good enough solution for some of the domains. But since ResponseOnEvent did not fulfill all the requirements a decision was made to implement a proprietary service instead in order to fulfill additional requirements. It is very hard to tell if diverging from the standard is worth the additional fulfilled requirements since following a standard could prove to simplify the integration of the service in the current system.

How can an event based service be realized in order to meet these requirements?

The suggested design of the service that was created during the thesis can be seen in appendix B. There were still some requirements that this service does not fulfill and some possible ways to solve this has been given in chapter 4.

Are there any problems connected to an implementation of the service?

Some problems were discovered both during the design of the proprietary service as well as during the implementation of the prototype. Different design decisions was made in order to solve these problems and others discovered later in the process has been mentioned and possible solution has been suggested. Below the different solutions and their consequenses as well as alternative ways of solving and implemneting the service will be discussed.

After having completed the prototype implementation a good insight into the pros and cons of event based services had been achieved. ResponseOnEvent and the proprietary service only allowed one trigger condition per serviceToRespondTo which later were seen as a big limitation to the flexibility of the service. The problems solved through this limitation was the mapping between an event response and an event logic in the client and to reduce the complexity for managing the different event logic in the server. It would be possible to introduce some type of logic identifier for each event logic, an id through which each event logic would be uniquely identified in the server. This number must then be communicated to the client when it sets up an event logic, for instance in the ROE initial response. This would make it possible to start, stop and clear individual event logic even when they are no longer uniquely identified through their serviceToResopndTo. The complexity introduced by this solution would be small but some changes would be required in the format of some messages. Also it would no longer be possible for the client application to map the event triggered responses to a certain event trigger condition if the application has set up multiple event logic with identical serviceToRespondTo.

If the above suggested id of the event logic could be included in the event responses this would solve the problem but it would change the format of the responses compared to the UDS or KWP standard. By following the standards and use the same format as ordinary diagnostic messages for the event responses, it might be possible to reuse some amount of code from the current system architecture when implementing the event based service. The amount of implementation work this would save is not clear tough. There is also a risk that by introducing new messages with a format similar to other messages, ECUs might confuse the messages with the original messages which in its turn could lead to problems and undefined behaviours. Therefore the ECUs' ability to distinguish the old messages from the new ones should first be tested before they are introduced in the system.

In the proprietary service the option to add time stamps to the responses was added which also diverges from ROE and the format of diagnostic messages defined in UDS and KWP. In conclusion, diverging from the ROE and the format of diagnostic messages would make the event service more flexible but might cause some additional implementation work. A suggestion of a new format for the event responses would be to use the ROE SID (0xC6) instead of the SID given by the serviceToRespondTo. This would reduce any eventual safety risks connected with the way that event responses were sorted out from the ordinary diagnostic responses in the client ECU. In the prototype implementation it was assumed that each response message that arrived to the client ECU that did not have a pending request, was an event response message. By using the ROE SID for the event triggered responses instead the event generated responses could be sorted out from the ordinary messages in a better way.

A suggested format of the event triggered responses that diverge from ROE and ordinary diagnostic format can be seen in table 5.1. Here the ordinary event triggered response message defined by ROE (here denoted by eventResponse) is packaged inside a message which uses the ROE SID. The logicIdentifier is the identifier given to the event logic in the server ECU as suggested above.

Data Byte	Parameter Name	Byte Value
#1	ResponseOnEvent Response SID	0xC6
#2	logicIdentifier	0x00-0xFF
#3	timeStamp_High Byte	0x00-0xFF
#4	timeStamp_Mid_High Byte	0x00-0xFF
#5	timeStamp_Mid_Low Byte	0x00-0xFF
#6	timeStamp_Low Byte	0x00-0xFF
#7	eventResponse = [	0x00-0xFF
:	:	:
#n	]	0x00-0xFF

Table 5.1: Suggestion of event triggered response containing time stamp and an event logic identifier.

One weakness with an event based service compared to sampling is that one can not know if there has occurred a problem in the server that has caused the event service to stop working. If no event responses are sent it could be because the service has not been triggered or that the service has stopped working. One way of solving this problem would be to have the client ask the server for its active event logic from time to time using the reportAllActivatedEvents service that is defined as a part of ROE. This could be a routine performed every time the ignition is turned on but there is no guarantee that this happens more than every 12th to 20th hour [42]. This could allow the service to be stopped for long periods of time before it is discovered that the service is not running properly. A better alternative would be to perform the checking for active event logic more frequently which of course gives rise to an increased network load which needs to be weighted against how important it is that the event service is running at all times.

5.2.1 Network Load

The most prominent benefit with an event based service is the possibility to perform vehicle diagnostics in a way that gives rise to a smaller amount of network load compared to sampling. Instead of sending a request and getting a response every second, the data will automatically be sent from the server when the service has been triggered. As a direct consequence of this the network load will be reduced as long as the service does not get triggered more often than the ordinary diagnostic service sampling rate. It is easy to understand how the event based service can gives rise to an increased network compared to sampling if the event occurs with a higher frequency than the sampling rate. This is illustrated in figure 5.1 and 5.2 where two different cases shows how the network load caused by an event based service depends on the number of times it gets triggered per time unit. The diagrams are based on fictitious data. The network load is measured in number of messages sent. The event based service starts at 10 messages since this is the largest amount of messages that it would take the proprietary service to setup and start its event logic. This is the case when the eventType onComparisionOfValue is combined with the serviceToRespond to ReadStatusOfDiagnosticTroubleCode. For each event, one message will be sent. The alternative to use the sampling method does not require any setup messages but for each sample two messages will be sent, one request and one response and this is independent of if the data of interest has changed or not.

Even if it is possible, it is very unlikely that the event service would give rise to an increased network load compared to the sampling as in the case shown in figure 5.2 since this would typically be considered a wrong way of using the service. The event service should not be setup to trigger on events that occur with a high frequency and thus it requires some knowledge of the person setting up the service or some type of restrictions in the service.

Due to the diagnostic services' low priority event triggered messages would not prevent more prioritised messages to be sent on the network but too many triggers from an event based service could cause problems inside the server ECU. Each ECU has an outgoing buffer into which each message is put before being sent on the bus. If the buffer would get flooded by diagnostic messages, the ECU would not be able to send other more critical messages which could lead to major faults [45]. This speaks in favor for some type of restriction of the number of messages that the event service should be allowed to generate.

All the communication on CAN today is deterministic. This means that all messages are sent with a predefined frequency and thus the bus load can be guaranteed to not exceed a certain value. The introduction of an event based service however introduces undeterminism. If multiple event logic is set up in a vehicle and they all start to generate messages at the same time this could cause a momentary overload of the bus. This is something that can happen even if the trigger conditions are set up to trigger on events that only occurs with a low frequency depending on how many instances of the service has been set up.

One way to prevent the above mentioned problems from happening is to not set up the service to trigger on events that happens too often. It would be possible to implement a function that determines what should be allowed to trigger on but it would most likely be a rather complex and memory intense function. One could also set a limit that prevents the service from sending more than a certain amount of messages every second. This solution is better since it would be very simple to implement and not increase the size of the service too much. These suggested solutions does not solve the problem connected to the undeterministic behaviour of an event based service. The only way to guarantee that multiple event logic does not cause a bus overload it to limit the amount of event logic that is allowed to be set up at the same time and for each of them, limit the amount of responses per second.

5.3 Future Work

During the thesis some areas were found that could be of interest to look further into. The will be summarised here.

ResponseOnEvent is just one way of defining an event based service for vehicle diagnostics. During the thesis, a brief investigation was made to find alternative services defined in other standards but non were found. A more thorough investigation could possibly reveal other alternatives to ResponseOnEvent which could prove interesting to look into.

The trigger conditions that were included in the final proprietary service were the same as in ResponseOnEvent. They are one of the key factor when talking about the flexibility of the service. Excluding a certain trigger condition could possibly render the service useless in some situations. It is therefore important that all the important trigger condition are found and included to maximize the flexibility of the service.

As it is now, the service runs every 10 milliseconds and checks for new fulfilled trigger conditions. This frequency is an upper limit of how fast a change can be detected inside the ECU. This frequency was not chosen based on any specific measurements and it should therefore be investigated further which frequency that is the most suitable to get a good enough resolution.

Possibly the detection of new fulfilled trigger conditions could be handled by function calls to the event service from the functions writing values to the ECU's memory. This was not investigated during the thesis but is an interesting way of increasing the resolution of the service and possibly reduce the amount of processing power required by it.

Different ECUs will be more attractive than others as the host of an event based service. Depending on which ECUs that are chosen, different constraints are placed on the service in the form of processing power and memory usage. In the thesis it was never investigated which ECUs that was the most interesting ones to implement the service on and this should therefore be done in order to decide how much resources the service should be allowed to consume. Maybe it will be discovered that the amount of available resources differ to such a degree that it could be beneficial to make different implementations of the service, some more extensive and others smaller.

Most of all it is interesting to look into how the motivation for an event based service measures against other services and if the motivation for the service is strong enough to be picked in front of other features that could be implemented using the same resources.

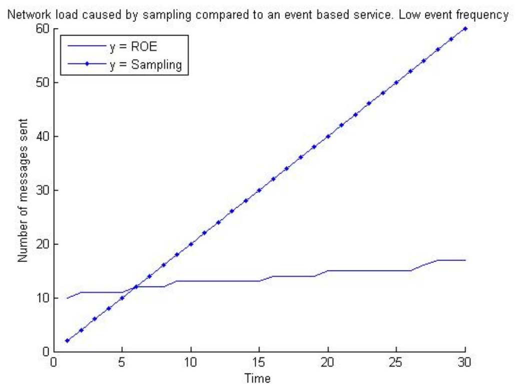

Figure 5.1: The network load for an event based service compared to sampling.

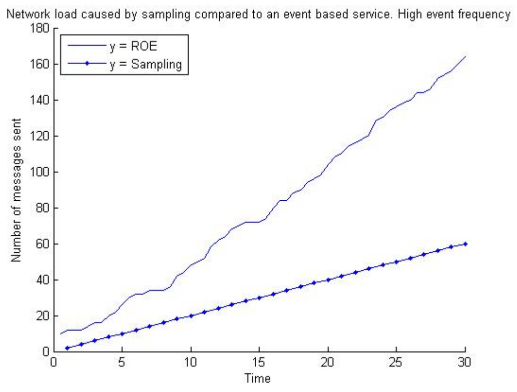

Figure 5.2: The network load for an event based service compared to sampling.

6 Conclusion

6.1 Conclusion

In this study it has been showed that there is a need for an event based service in the area of vehicle diagnostics. It has also been shown that there are some complications connected with a realization of such a service. One of the biggest complications is the fact that most embedded systems into which the service should be implemented have a very limited amount of available memory and CPU-power. An event based service also requires the server to send diagnostic responses without first getting a diagnostic request which can require a reconstruction of the current software architecture. But with an event based service comes great benefits. Beyond the possibility to carry out diagnostics in a more efficient way compared to sampling, there is the possibility to improve the diagnostics of more complex functions due to the flexibility of an event based service.

How does ResponseOnEvent defined by the UDS-standard meet existing needs of diagnostics on heavy duty vehicles?

As pointed out by the elicitation of requirements, the ResponseOnEvent service is not sufficient in order to support all the requirements found in the domains. The interviews did however indicate that the three domains investigated during the thesis would benefit from an event based service similar to ResponseOnEvent.

How can an event based service be realized in order to meet these requirements?

The proprietary service taken forth as an alternative to ResponseOnEvent supported four additional requirements and an additional diagnostic service which could be used together with the proprietary service. The proprietary service was made to follow ResponseOnEvent as much as possible to simplify the design and speed up the process. Possibly there are other alternative ways of how an event based service could be designed.

Are there any problems connected to an implementation of the service?

The prototype developed showed on the need for a reconstruction of the software architecture as well as some decisions that needed to be made concerning the behaviour of the service.The design and implementation of the service should aim to keep the service memory and CPU-efficient so that it can become accessible to more ECUs even though they have a very limited amount of free memory and CPU-power. But even if the ECU has enough free resources, the event based service will be competing for these against other future functionality which might be deemed more critical.

The proprietary service shows that a service can be implemented using the memory that is available in the TMS. The service did also manage use less than 1% of the processing power. The processing power needed could also have been regulated by decreasing or increasing the frequency of how often the service runs its loop inside the ECU.

Another important finding is the importance of trigger conditions. As described under section 4.5.2 as well as 5.2.1 the selection of trigger conditions is essential for the event based service performance. By allowing multiple trigger conditions for an event, it would enable more complex diagnostic. However the client ECU will not be able to conclude which condition that has been triggered when it receives the event generated response message if no extension is made of the response messages. Furthermore by selecting high frequent trigger conditions there is a risk of flooding the network with diagnostic data which is why the service should only be used together with event conditions that do not become fulfilled too often. This could easily be solved by limiting the amount of responses that the service can generate every second.

Many of the problems found during the thesis can be solved in different ways. It is however hard to decide which way is the best since it is still unclear exactly how and when the event based service will be used. To make the right limitations is crucial to make the service as flexible and useful as possible.

Appendices

A Appendix: Interpretation of ResponseOnEvent

A.1 Interpretation of ROE

The uncertainties that were found in the UDS's definition of ROE are summarized below together with the chosen interpretation (the page numbers refers to the pages in the standard).

1. Page 76: When an event occurs the serviceToRespondTo shall be executed, and if conditions are not correct a negative response message with the appropriate negative response code shall be sent.
   -It is unclear which conditions the standard is referring to here.

   Interpretation
   Conditions is assumed to refer to the servers ability to execute the serviceToRespondTo. The server might not support the serviecToRespondTo in the current session or without additional security access.

2. Page 76: An implementation hint in the standard states that a request to move to any non-default session shall stop all instances of the ROE services in that client. On returning to the default session all instances of ROE that were stopped shall be re-activated.
   -If the server stops its instances of ROE due to a request to move to a non-default session from one of its clients, shall the other clients with active instances of ROE in that ECU be notified? It might be beneficial that they know that their instance of the service has been stopped.

   Interpretation
   Nothing was mentioned about this in the standard and thus chosen to be a possible extension. Since an ECU per default is in the default session and only occasionally moves to another session due to a request from for instance a mechanic connected to the vehicle using a PC, the number of event triggers that would be missed during that time is limited. If the event logic also is re-activated upon returning to the default session there should be no need for such notifications.

3. Page 76: Concerning the same implementation hint above, shall the event services activated in any non-default session be re-activated upon returning to that session?

Interpretation
Reactivation of event logic upon returning to for instance the programming session could complicate the process of re-programming the ECU for example. If the event logic is being continuously triggered this can block the access to the ECU for re-programming. This could easily be solved by sending a stop request to the ECU but no need for this type of automatic reactivation was seen and thus it was decided that ROE should only reactivate event logic when returning to the default session.

4. Page 76: The start and stop sub functions for ROE shall control all the initialized ROE services.
   -Does this include services that were initialized in other sessions or just the ones initialized in the current session?

   Interpretation
   Since nothing else was stated, it was decided that the start and stop function should activate respectively deactivate all the initialized ROE services, independent of in which session they were initialized. This might cause error messages to be sent at certain events instead of the intended serviceToRespondTo. This since the serviceToRespondTo might not be supported in the current session.

5. Page 76: It is stated that "the server shall respond to the client which has set up the event logic and has started the ROE events".
   -It is unclear where the event messages will be sent if one ECU sets up the event logic and another one starts it.

   Interpretation
   The responses to events shall be sent to the client who set up the event. The client who started the event logic shall always receive an initial response message if the suppressPosRspMsgIndicationBit was not set to "TRUE" ('1').

6. Page 76: "Multiple events shall be signaled in the order of their occurrence."
   -Does this apply for all active services in all ECUs or all active services in one ECU or for each service individually in each ECU?

   Interpretation
   Since it seemed impossible to order all events in a whole network it was assumed that the order only applied for event in each ECU individually.

7. Page 77: If an instance of the service is set up with infinite eventWindowTime and storage status set to storeEvent, the service shall be automatically started upon power up.
   -Does this apply to services initiated in any session?

   Interpretation
   Since nothing else was stated, the assumption was made that each event with an infinite eventWindowTime and storage status set to storeEvent should be started on power up independent of in which session they were initialized.

8. Page 77: In the UDS standard there is a list of services which the standard recommends as the only services that should be used as the serviceToRespondTo.
   -What is the reason only these services should be used?
   -It could be beneficial to be able to use other services such as ReadMemoryByAddress. Could this service be used together with ROE?

Interpretation
The responses that the recommended services could generate were not limited to a certain size, so the reason for the recommendation was not to guarantee that the response could be generated within the execution time of the service. No reason was seen to not make it possible to use other services together with ROE, but it was assumed that this would be considered to go against the standard.

9. Page 79: An instance of the ROE service with the storage status storeEvent, shall be saved in the ECU until it is cleared using clearResponseOnEvent or overwritten by a new ROE setup event logic request of the same category.
   -It is unclear what category refers to here. Category is not mentioned again in the UDS standard 14229-1_2013.

   Interpretation
   It was assumed that the category meant serviceToRespondTo, since this would solve the problem with identical responses to different trigger conditions.

10. Page 79: The subfunction onDTCStatusChange comes with an implementation hint: "A server resident DTC count algorithm shall count the number of DTCs satisfying the client defined DTCStatusMask at a certain periodic rate", "If the count is different from that which was calculated on the previous execution, the client shall generate the event that causes the execution of the serviceToRespondTo. The last count shall then be stored as a reference for the next calculation."
    -This would cause events to trigger of not only new DTCs that matches the mask but also triggered by old DTCs that change state so that they no longer match the status mask. This stands in conflict with the defenition of the service which stated that it should trigger on new DTCs.

    Interpretation
    Scania had previously contacted the ones responsible for writing the ISO standard and confirmed that the service should be triggered also by DTCs that change state.

11. The same implementation hint mentioned above would sometimes cause the service not to trigger even if a DTC's status would changed, as long as the total amount of DTCs matching the status mask remains the same. It would however be of interest to trigger the event even at these occasions.
    -Should the word "count" in the hint be interpreted as counting the amount of DTCs for which the comparison results in a different value compared to previous calculation? This would not be consistent with the trigger condition "If the count is different from that which was calculated on the previous execution, the client shall generate the event", since the event should be triggered even if the number of DTCs that change status between the calculations happens to add up to the same amount as the last time.

    Interpretation
    Scania had previously contacted the ones responsible for writing the ISO standard and confirmed that it would be better if the event is triggered by counting the number of DTCs for which the comparison resulted in a different result this iteration compared to the previous iteration and if the sum is greater than zero, the event should be triggered.

12. In the above citations from the standard, the word "client" is used in the following text "the client shall generate the event that causes the execution of the serviceToRespondTo".
    -If generate the event means to trigger the execution of the serviceToRespondTo, should it not be the server who generates the event and not the client? Is this a printing error?

Interpretation
This was assumed to be a printing error.

13. It is not stated if a client can start and stop another clients event logic in a server, and who will get the "initial" response and who will get the serviceToRespondTo responses.

    Interpretation
    To make the service more flexible and take height for future needs it was assumed that it should be possible for a client to activate other clients event logic in servers. The initial response should be sent to the client who activated the service and the event triggered responses should be sent to the client who set up the event logic.

14. Page 88: The response message to sub-function startResponseOnEvent will include a field called eventType. In the example in table 112 this is set to onDTCStatusChange which was the eventType of the event logic that was set up in the same example. Since startResponseOnEvent shall start all the initialized event logic in the ECU, what should the responseMessage look like if multiple event logic are activated by the startResponseOnEvent request? Should multiple responses be sent, one for each started service?

    Interpretation
    It was assumed that any of the eventTypes of the activated event logic could be used in the eventType filed and that only one response should be sent. This did not clash with the standard but the standard was very unclear about this. A better solution that would diverge from the standard would be to set the eventType of the response equal to the eventType of the request, as in all the other cases when a client communicates with a server.

15. Page 88: The response message to sub-function clear and stop responseOnEvent contains a field called numberOfIdentifiedEvents. It is unclear what this field should contain in the case that all the event logic in an ECU is cleared or stopped. Should it be the sum if all the different event logic? What should it contain when the clear sub-function is used to clear non-active event logic? Should the count from the last time the event logic was active be returned?

    Interpretation
    It was assumed that the responses to the stop and clear sub-functions should contain the sum for all active event logic since this could be valuable information.

B Proprietary Service

Here a proprietary service designed as an modification of the service ResponseOnEvent found in ISO 14229-1:2013 will be described. Where any uncertainties might arise, see ResponseOnEvent for clarification.

B.1 Event Service setup

ServiceToRespondTo

The client sets up the service in another ECU by sending a responseOnEvent request message to that ECU. For a description of the message see table B.3. The request message contains the event logic, which consists of a trigger condition (eventType) and a diagnostic service (serviceToRespondTo) that the server shall respond to when the trigger condition becomes fulfilled. The diagnostic services supported as serviceToRespondTo are listed in table B.1.

The different eventTypes or sub-functions that can be used as trigger conditions are listed in table B.9 and B.10. The table also contains sub-functions which are used for manipulating the service in ways such as setting up, starting and stopping the service.

It is only possible to set up one trigger condition for each serviceToRespondTo in an ECU. This will prevent the client from setting up identical serviceToRespondTo combined with different trigger conditions.

Overwrite

The initialization of new event logic replaces the old logic in the server for that specific serviceToRespondTo.

Initial Response

The server will check the following parameters in the ResponseOnEvent request:

- sub-function [eventType]
- eventWindowTime

- eventTypeRecord (eventTypeParameter #1-#m)

The server shall send a negative response message with the NRC `0x31` if some of the parameters are invalid. See table B.8 for a description of the error message.

If the request message was used for setting up new event logic (sub-function type equal to setup) and all the parameters were valid, a initial positive response is sent. See table B.5 for description of the message.

Storage State

The storage state bit is used to indicate that an event logic shall be activated when the ECU powers up. An event logic is automatically activate on power up if storage state is set to “storeEvent” (‘1’).

The storage state bit can only be set to "storeEvent" (‘1’) if the event window time is infinite.
*Time Stamps of Responses
When setting up an event logic, the client can define in the responseOnEvent request message to have his responses to events being time stamped. This is indicated by setting the timeStamp bit to “TRUE” (‘1’). This will add one or more extra bytes to the response messages that will be sent when the specified event occurs. The extra bytes should contain a time stamp for when the event logic was triggered. If time stamps are used this will cause the response messages to differ from their KWP definitions. An example is to use the first byte for hours, the second for minutes, third for seconds and fourth for hundredth of a second.

ServiceToRespondTo	Request SID
ReadStatusOfDiagnosticTroubleCode	0x17
ReadDataByCommonIdentifier	0x22
ReadMemoryByAddress	0x23
InputOutputControlByCommonIdentifier	0x2F
StartRoutineByLocalIdentifier	0x31
StopRoutineByLocalIdentifier	0x32
RequestRoutineResultsByLocalIdentifier	0x33

Table B.1: Supported KWP services as serviceToRespondTo

B.2 Starting an event logic

To start all the initialized event logic in an ECU the message defined in table B.4 is used, with the eventType equal to startResponseOnEvent (`0x05`). The format is equal to the first three bytes in the responseOnEvent request message, see table B.3

It is possible to start a single event logic using the same message format but using the eventType startSingleResponseOnEventXX where XX is substituted with the eventType which should be started. The corresponding hexadecimal number is found in table B.9 and B.10. Table B.2 shows an example request message for starting a single event logic with the serviceToRespondTo equal to ReadDataByCommonIdentifier. The positive response message to the startResponseOnEvent or startSingleResponseOnEventXX request is defined by table B.7 and is sent only if the suppressPosRresponseMessageIndicationBit was set to “False” (‘0’) in the request message.

Data Byte	Parameter Name	Byte Value
#1	ResponseOnEvent SID	0x86
#2	eventType = startSingleResponseOnEventReadDataByCommonIdentifier	0x24
#3	eventWindowTime (will not be evaluated)	0x00

Table B.2: Message for starting a single event logic with the serviceToRespondTo equal to ReadDataByCommonIdentifier

The "suppressPosResponseMessageIndicattionBit" should only be set to "TRUE" ('1') if the sub-function is any of the following: start, stop and clear response on event. Here start and stop refers to any of the start and stop sub-functions listed in table B.9 and B.10.

If the specified event logic to be activated was not initialized or already active, an error message with the NRC "conditionsNotCorrect" (`0x22`) will be sent. The format is given by table B.8.

B.3 Stopping an event logic

It is possible to stop (deactivate) a single event logic using the corresponding stopSingleResponseOnEventXX sub-function. XX is replaced by for the corresponding serviceToRespondTo.

It is possible to stop all the event logic in an ECU using the stopResponseOnEvent sub-function (`0x00`).

A positive response message will be sent if the suppressPosRresponseMessageIndicationBit was set to "False" ('0') in the request message. The positive response message has the same format as in the case of starting event logic, see B.7.

If stopSingleResponseOnEventXX was used and the specified event logic to be stopped was not active, a NRC "conditionsNotCorrect" (`0x22`) will be sent.

B.4 Clearing Event Logics

It is possible to clear a single event logic using the clearSingleResponseOnEventXX sub-function. It is also possible to clear all the event logic in an ECU using the clearResponseOnEvent sub-function (`0x06`).

If an event logic was cleared using the sub-function clearSingleResponseOnEventXX, the server shall respond with a positive response only if the suppressPosRresponseMessageIndicationBit was set to "False" ('0').

If clearSingleResponseOnEventXX was used and the specified event to be cleared was not initialized, a NRC "conditionsNotCorrect" (`0x22`) should be sent.

The server shall always respond with a positive response if the sub-functiuon clearResponseOnEvent was used and the suppressPosRresponseMessageIndicationBit was set to "False" ('0').

A cleared event logic is permanently removed from the server. An active event logic that is cleared shall be removed and any pending events shall be discarded.

B.5 Sub-functions (EventTypes)

The sub-functions or eventTypes supported by the service are summarized in tabel B.9-B.10. Of them the following are used for setting up new event logics: onDTCStatusChange, onTimerInterrupt, onChangeOfDataIdentifier and onComparisonOfValues. They are all labeled with the sub-function type "setup". The others are used for controlling the service in ways such as clearing event logic, starting event logic or stopping event logic. They are labled with the sub-function type "control".

The response to reportActivatedEvents is extended with two bytes per reported event logic compared to the UDS standard 14229-1:2013. The additional bytes contains the address of the ECU to whom the serviceToRespondTo is sent and the setting regarding time stamps. The response is shown in table B.6.

B.6 Triggering of an Event

Events shall be signaled in the order of their occurrence in each ECU individually.

When event logic is triggered the serviceToRespondTo shall be executed. If there is an error in the serviceToRespondTo or the serviceToRespondTo is not supported in the current session the server shall send a negative response with appropriate error code to the client.

In case additional events occur while another event is being processed, the later events will be stockpiled and processed when the first event is done.

When defining the serviceToRespondTo note must be taken to the execution time of the service. If the service takes to long to execute it might affect the performance of the whole system. For instance, when using ReadMemoryByAddress as the serviceToRespondTo, the parameter MemorySize defining how much data that should be communicated, should not be taken too large since this might cause the execution of the service to take too long.

B.7 Sessions

Event logic can be set up and activated in any session.

The event logic will be deactivated when the session is changed in the ECU from default to a non-default session. Upon returning to the default session that event logic will be started again. No start response message shall be sent to the client ECU.

If an ECU is requested to move to the default session from the default session the event logic shall not be deactivated.

If an ECU is requested to move to a non-default session from any session the event logic will be deactivated.

B.8 Window time frame

The server shall return a ″final″ response to the client only when the event's finite event window time has elapsed. No final response shall be sent if the event logic was stopped by any other means. The format of this final response message is the same as for the initial response message shown in table B.5.

When an event window time frame has elapsed the event logic shall be stopped and all pending events shall be discarded.

B.9 MultiClients

The response to all the start request messages will be sent to the client who sent the start request, independent of to whom the serviceToRespondTo responses will be sent.

Each client can activate the event logic within an ECU regardless of the client who set up the event logic.

Each client can deactivate the event logic within an ECU regardless of the client who set up or activated the event logic.

Each client can initialize new event logic regardless of the client that previously set up the event logic.

Any client can clear any event logic in a server. (To prevent clients from being permanently blocked by a lost event logic)

B.10 Tables

Data Byte	Parameter Name	Byte Value
#1	ResponseOnEvent Request SID	0x86
#2	sub-function [eventType] = Bit #1-6: eventType Bit #7: storageState Bit #8: suppressPosRspMsgIndicationBit	0x00-0xFF
#3	eventWindowTime	0x00-0xFF
#4	responseInformation = Bit #1: timeStamp Bit #2-8: vehicle Manufacture Specific	0x00-0xFF
	eventTypeRecord[] = [	
#5	eventTypeParameter_1	0x00-0xFF
..	..	..
#5+(m-1)	eventTypeParameter_m]	0x00-0xFF
	serviceToRespondToRecord[] = [	
#n-(r-1)-1	serviceId	0x00-0xFF
#n-(r-1)	serviceParameter_1	0x00-0xFF
..	..	..
#n	serviceParameter_r]	0x00-0xFF

Table B.3: ResponseOnEvent request message used for the setup sub-functions

Data Byte	Parameter Name	Byte Value
#1	ResponseOnEvent SID	0x86
#2	eventTypeRecord[eventType] = Bit #1-6: eventType Bit #7: storageState (will not be evaluated) Bit #8: suppressPosRspMsgIndicationBit	0x00-0xFF
#3	eventWindowTime (will not be evaluated)	0x00

Table B.4: ResponseOnEvent request used for the control sub-functions

Data Byte	Parameter Name	Byte Value
#1	ResponseOnEvent Response SID	0xC6
#2	sub-function [eventType] = Bit #1-6: eventType Bit #7: storageState Bit #8: suppressPosRspMsgIndicationBit	0x00-0xFF
#3	numberOfIdentifiedEvents	0x00-0xFF
#4	eventWindowTime	0x00-0xFF
#5	responseInformation = Bit #1: timeStamp Bit #2-8: vehicle Manufacture Specific	0x00-0xFF
 #6 .. #6+(m-1)	eventTypeRecord[] = [eventTypeParameter_1 .. eventTypeParameter_m]	 0x00-0xFF .. 0x00-0xFF
 #n-(r-1)-1 #n-(r-1) .. #n	serviceToRespondToRecord[] = [serviceId serviceParameter_1 .. serviceParameter_r]	 0x00-0xFF 0x00-0xFF .. 0x00-0xFF

Table B.5: ResponseOnEvent initial positive response message for all eventTypes except ReportActivatedEvents (0x04)

Data Byte	Parameter Name	Byte Value
#1	ResponseOnEvent Response SID	0xC6
#2	eventType = ReportActivatedEvents	0x04
#3	numberOfActivatedEvents	0x00-0xFF
#4	eventTypeOfActivatedEvent#1	0x00-0xFF
#5	eventTypeWindowTime#1	0x00-0xFF
#6	responseInformation = Bit #1: timeStamp Bit #2-8: Future needs	0x00-0xFF
#7	clientAddress	0x00-0xFF
  #8 .. #8+(m-1)	eventTypeRecord#1[] = [eventTypeParameter_1 .. eventTypeParameter_m]	 0x00-0xFF .. 0x00-0xFF
  #p-(o-1)-1 #p-(o-1) .. #p	serviceToRespondToRecord#1[] = [serviceId serviceParameter_1 .. serviceParameter_o]	 0x00-0xFF 0x00-0xFF .. 0x00-0xFF
:	:	:
#n-(r-1)-4-(q-1)	eventTypeOfActivatedEvent#k	0x00-0xFF
#n-(r-1)-3-(q-1)	eventTypeWindowTime#k	0x00-0xFF
#n-(r-1)-2-(q-1)	responseInformation = Bit #1: timeStamp Bit #2-8: Future needs	0x00-0xFF
#n-(r-1)-1-(q-1)	clientAddress	0x00-0xFF

 #n-(r-1)-(q-1)
 :
 #n-(r-1)
	eventTypeRecord#k[] = [
	 eventTypeParameter_1
	 ..
	 eventTypeParameter_q]
		0x00-0xFF
		 ..
		 0x00-0xFF

  #n-(r-1)+1 #n-(r-1)+2 .. #n+2	serviceToRespondToRecord#k[] = [serviceId serviceParameter_1 .. serviceParameter_r]	 0x00-0xFF 0x00-0xFF .. 0x00-0xFF

Table B.6: ResponseOnEvent positive response message for ReportActivatedEvents (0x04)

Data Byte	Parameter Name	Byte Value
#1	ResponseOnEvent SID	0xC6
#2	eventType	0x00-0x7F
#3	numberOfIdentifiedEvents	0x00-0xFF
#4	eventWindowTime	0x00-0xFF

Table B.7: ResponseOnEvent positive response message for all control sub-functions except reportActivatedEvents

Data Byte	Parameter Name	Byte Value
#1	Negative Response SID	0x7F
#2	Request SID	0x86
#3	NegativeResponseCode (NRC)	0x00-0xFF

Table B.8: Negative response message

Bits 5-0 Value	Description	sub-function type
0x00	**stopResponseOnEvent** Used to stop ResponseOnEvent in the server. The event logic is not cleared from the server's memory. Length of eventTypeRecord: 0 byte	control
0x01	**onDTCStatusChange** Defines the event when a new DTC is detected that matches the defined DTCStatusMask supplied in the eventTypeRecord of the message Length of eventTypeRecord: 1 byte	setup
0x02	**onTimerInterrupt** Defines the event as a timer interrupt. Every time the timer elapses an event is triggered. Length of eventTypeRecord: 1 byte	setup
0x03	**onChangeOfDataIdentfier** Defines the event as an internal data record changes its value. Length of eventTypeRecord: 2 byte	setup
0x04	**reportActivatedEvents** Is used to check which instances of the responseOnEvent service that are active in the server Length of eventTypeRecord: 0 byte	control
0x05	**startResponseOnEvent** Used to start ResponseOnEvent in the server. Length of eventTypeRecord: 0 byte	control
0x06	**clearResponseOnEvent** Used to clear the event logic that has been set up in the server. Length of eventTypeRecord: 0 byte	control
0x07	**onComparisonOfValues** Used to define event logic that compares data defined by a specified dataIdentifier and a given value using one of the following specified comparison operator: >, <, =, <>. The event occurs if the comparison is positive. Length of eventTypeRecord: 10 byte	setup
0x20	**stopSingleResponseOnEventReadDataByCommonIdentifier** Used to stop a single ResponseOnEvent in the server with the service ReadDataByCommonIdentifier as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x21	**stopSingleResponseOnEventReadDTCInformation** Used to stop a single ResponseOnEvent in the server with the service ReadDTCInformation as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x22	**stopSingleResponseOnEventRoutineControl** Used to stop a single ResponseOnEvent in the server with the service RoutineControl as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x23	**stopSingleResponseOnEventInputOutputControlByIdentifier** Used to stop a single ResponseOnEvent in the server with the service InputOutputControlByIdentifier as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control

Table B.9: Supported sub-functions (eventTypes)

0x24	**startSingleResponseOnEventReadDataByCommonIdentifier** Used to start a single ResponseOnEvent in the server with the service ReadDataByCommonIdentifier as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x25	**startSingleResponseOnEventReadDTCInformation** Used to start a single ResponseOnEvent in the server with the service ReadDTCInformation as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x26	**startSingleResponseOnEventRoutineControl** Used to start a single ResponseOnEvent in the server with the service RoutineControl as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x27	**startSingleResponseOnEventInputOutputControlByIdentifier** Used to start a single ResponseOnEvent in the server with the service InputOutputControlByIdentifier as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x28	**clearSingleResponseOnEventReadDataByCommonIdentifier** Used to clear a single ResponseOnEvent in the server with the service ReadDataByCommonIdentifier as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x29	**clearSingleResponseOnEventReadDTCInformation** Used to clear a single ResponseOnEvent in the server with the service ReadDTCInformation as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x2A	**clearSingleResponseOnEventRoutineControl** Used to clear a single ResponseOnEvent in the server with the service RoutineControl as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x2B	**clearSingleResponseOnEventInputOutputControlByIdentifier** Used to clear a single ResponseOnEvent in the server with the service InputOutputControlByIdentifier as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x2C	**stopSingleResponseOnEventReadMemoryByAddress** Used to stop a single ResponseOnEvent in the server with the service ReadMemoryByAddress as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x2D	**startSingleResponseOnEventReadMemoryByAddress** Used to start a single ResponseOnEvent in the server with the service ReadMemoryByAddress as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x2E	**clearSingleResponseOnEventReadMemoryByAddress** Used to clear a single ResponseOnEvent in the server with the service ReadMemoryByAddress as serviceToRespondTo. Length of eventTypeRecord: 0 byte	control
0x2F	**vehicleManufactureSpecific**	

Table B.10: Continuation: Supported sub-functions (eventTypes)

C Controller Area Network

This appendix describes the important components of the Controller Area Network (CAN) in a more detailed way compared to chapter 2.4.

C.1 Protocol and architecture

A Controller Area Network consist of one or more CAN buses that connect two or more ECUs that implement the CAN protocol. Each ECU connected to the same bus will be able to read all the messages on the bus [25]. But sometimes it can be beneficial to direct messages to certain ECUs for instance when certain values from the ECU should be read or overwritten. Therefore the CAN protocol specifies a certain field in the CAN messages called identifier which can be used by the ECUs to filter the communication so that the ECU only processes certain messages. [30] [15]

In vehicle CAN often consists of several main buses with the purpose of separating critical components from less critical. Figure 2.6 shows an instance of Scania's CAN network with three main buses. The rightmost bus contains the most critical systems such as engine management systems, while the middle contains the not as critical systems such as alarm system. The leftmost bus contains non-critical systems such as climate control. The three buses are joined together by a coordinator gateway ECU called COO or Coordinator. The data transmission rates vary between the different buses depending on how critical the systems on that bus are. CAN allows data rates from 20kbit/s up to 1Mbit/s [14]. Due to its light protocol management, the deterministic resolution and its error detection and retransmission CAN is a good way for embedded systems to communicate. [15][18][23]

Each ECU supports a set of different diagnostic messages and each message has its own unique identifier. The identifier is determined in advance by the developers to make sure that all message identifiers are unique within the system.

CAN allows only one ECU at a time to use the bus for sending messages. Each message that is sent on the bus is called a data frame and it is divided into different fields. There are two types of CAN messages, the ones using the standard version identifier and the ones using the extended version identifier. In figure C.1 a CAN frame using the extended version is depicted. It is enough to say that the standard version is similar to the extended version but uses fewer bits for the identifier field. The extended version is the one that the C300 and the TMS use and therefore the one that will be focused on here. In table C.1 the different fields are described. [5]

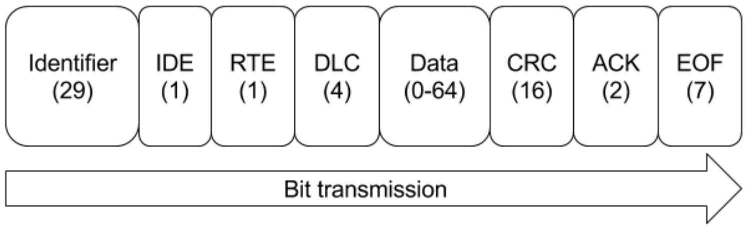

Figure C.1: Simplified structure of an extended CAN data frame

Bits	Abbreviation	Description
29	Identifier	This field contains 29 bits (extended version). It serves two purposes. The identifier specifies the type of the message and the target address. It is also used to determine which message that has the highest priority, i.e. which message should be sent on the bus when multiple hosts are trying to send their message at the same time. [5][10] How the different bits of the identifier should be interpreted is defined in the standard J1939 [31].
1	IDE	A single bit that indicates the format of the identifier. It is set to '0' if a standard version (11 bits) are used and '1' if the extended (29 bits) version is used.
1	RTR	The remote transmission request bit is set to '0' when information should be fetched from another node. Otherwise it is set to 1.
4	DLC	4 bits that tells the length of the data field.
0-64	Data	The payload. The first byte defined the number of following significant data bytes in the message.
16	CRC	To make CAN reliable, a CRC (cyclic redundancy field) field is used so that the receiving host can determine if the received payload was corrupted or not during the transmission. If a host receives a corrupted message, the protocol enables the host to send an error flag on the bus and thus resolve the issue.
1	ACK	The bit is used in the response message and is set to '0' to indicate if the receiver received an error free message. If an error was detected the bit is set to '1' and the sending node re-transmits the original message.
7	EOF	The seven End of Frame bits indicated the end of the frame and must be set to '1's.

Table C.1: Fields in a CAN frame with extended identifier [5] [10]

C.2 Identifier

A simplified version of the 29-bit identifier is shown in figure C.2. The different fields are described in table C.2. [26]

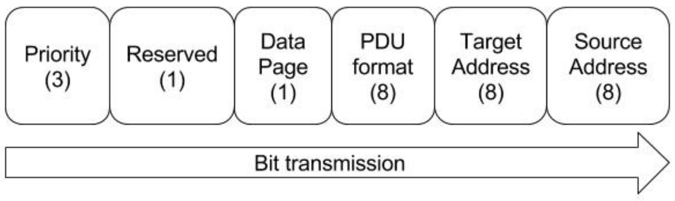

Figure C.2: Simplified figure of a 29-bit identifier according to J1939

Bits	Abbreviation	Description
3	Priority	Defines the priority of the frame.
1	Reserved	Reserved for future needs. Will be set to ’0’.
1	Data Page	Should be set to ’0’.
8	PDU Format (PF)	Defines if the message should be broadcast (0xDB) or sent to a specific address (0xDA).
8	PDU Specific (PS)	Contains the address of the target ECU or the broadcast address.
8	Source Address	The sending ECU’s address.

Table C.2: The important fields in the extended identifier version [31]

D Diagnostic Domain Analysis

For studies where the research goals are of a qualitative nature, it is generally appropriate to rely on qualitative measures. This section describes the qualitative method that was used to investigate the different domains and whether an event based service could improve their diagnostics.

D.1 Semi-structured interview

Interviewing people provides an opportunity to partake in their opinions, thoughts and knowing. For the interviews to be as efficient as possible it is important that the interviewees feel comfortable, so that they are willing to share their experiences.[12]

Semi-structured interviews are a combination of structured and unstructured interviews, which also can be referred to as focused interviews. These types of interviews use open-ended questions with the intent to elicit unexpected information. Using open-ended questions means that the questions are formulated in a structured way that allows follow up questions [12].

By having few questions asked with respect to a larger perspective brings the conversation to a more natural state, which then causes the interviewee himself to some extent control the order of the interview. The purpose is to get the person to tell you as much as possible without him being lead or boxed in by the questions [20]. Using follow up questions during the interviews are an important part of the open-ended questions for validating the information, making sure that interpretations are limited.
Before the interviews are conducted, it is important to select which subjects to interview carefully in order to achieve the best result [6]. The subject needs to be knowledgeable within the area and well informed of the purpose of the interview. Selecting a subject higher up in the hierarchy usually gives a broader perspective, leading to more generalized result. For more technical accurate result it is more suitable to chose a subject closer to the tool or product used [3].

Questions	Purpose
Describe what you do?	Obtain background of the person and the area of domain.
Describe why you do what you do?	Identify the purpose and goal.
Describe a typical situation when your area is useful?	Identify in which context it is useful.
What are the tools/services used?	What is the situation today.
Describe how they differ from each other?	What features are provided.
Describe a situation when the current diagnostic methods are not working.	What the restrictions are.
Describe when an event based service would be useful to you?	Potential features.
Do you see any potential limitations with an event based service?	Potential risk.

Table D.1: Example of questions used during the thesis

D.2 Analyzing qualitative interview data

Analyzing qualitative data commonly includes five stages such as data preparation, data acquaintance, data interpretation, data verification and data presentation. The nature of a qualitative method is of repetitive character, meaning that the researcher must go back and forth between the steps. The process of going back and forth is a necessity for comparing aspects or find recurring themes [16].

During the interviews notes and memos are made on about how to categorize the data. The difficulties with semi-structured interviews lies in how to analyze the data. Memos and notes are ideas and theories on how proceed with the analyzes at the initial phase and the transcripts are read through checking for any inconsistencies, preparing the data for the following stages. Data acquaintance are made by reading the transcripts and making notes on general themes within the transcript. The transcripts are read through repeatably and as many categories as necessary are generated. This stage is known as open coding. [1] [16] [3]

When interpreting the data key decisions are necessary in order to reach generalized conclusions. This involves prioritizing certain parts of the data, meaning that attentions is directed to certain parts while other is left to the side. The list of generated categories with similar themes are grouped together. The purpose is to reduce the number of categories by merging similar themes into wider categories. [8] [1] [16]

Validating the data is important due to the nature of the method. The method described involves risk of taking the meaning out of its context. For validating the method and the several steps involved there is need of more than one person. Categories, prioritizing the data and making key decisions in order to reach generalized decision can be compared if there are several people involved. [16] [8]

E TMS UML Diagrams

This appendix contains UML diagrams describing the communication between different parts of the event based service in the server application. This appendix addresses the interested reader who have a basic understanding of Scania's software architecture.

E.1 Setup

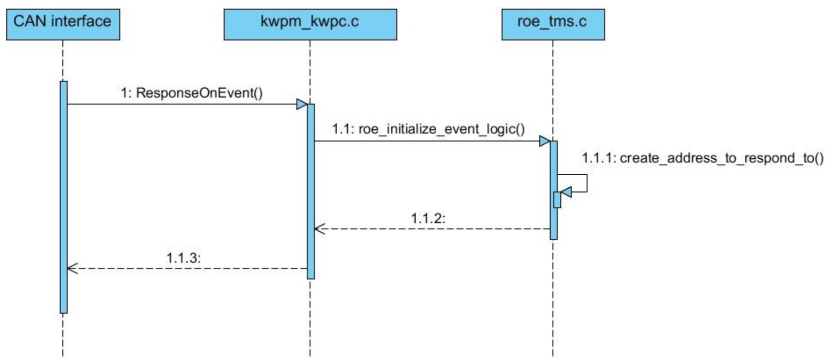
CAN interface
kwpm_kwpc.c
roe_tms.c
1: ResponseOnEvent()
1.1: roe_initialize_event_logic()
1.1.1: create_address_to_respond_to()
1.1.2:
1.1.3:

Figure E.1: A sequence diagram describing the process for setting up an event logic. The CAN interface calls the ResponseOnEvent method defined in kwpm_pwpc.c which interprets the content of the incoming CAN message and in its turn calls the method for setting up the event logic. This method is located in the roe_tms.c file and it manages the setup of the event logic.

E.2 Start

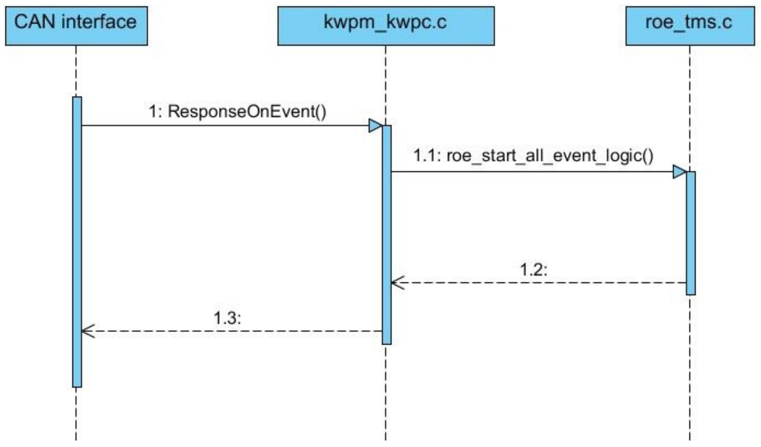

Figure E.2: A sequence diagram describing the process for starting an event logic. The CAN interface calls the ResponseOnEvent method defined in kwpm_pwpc.c which interprets the content of the incoming CAN message and in its turn calls the method for starting the event logic.

E.3 Check for event triggers

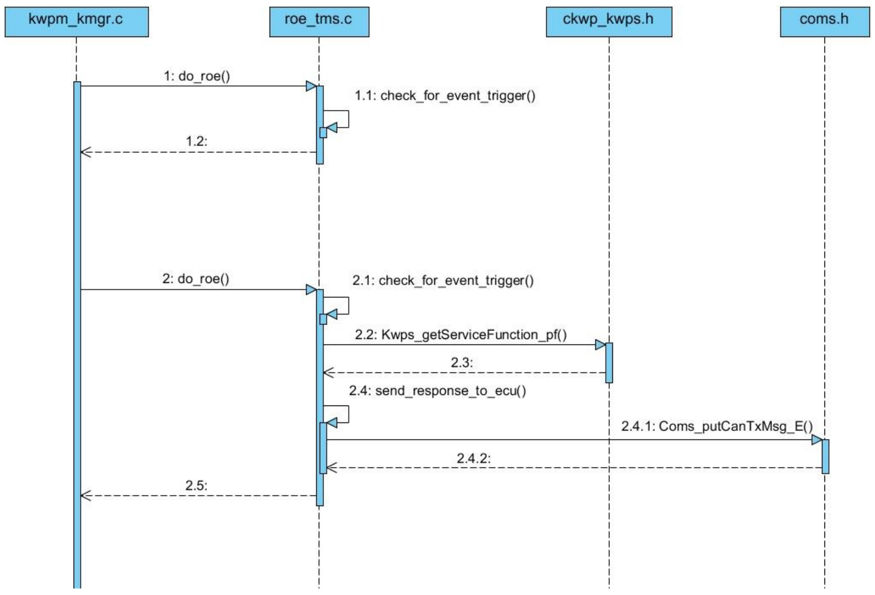
kwpm_kmgr.c
roe_tms.c
ckwp_kwps.h
coms.h
1: do_roe()
1.1: check_for_event_trigger()
1.2:
2: do_roe()
2.1: check_for_event_trigger()
2.2: Kwps_getServiceFunction_pf()
2.3:
2.4: send_response_to_ecu()
2.4.1: Coms_putCanTxMsg_E()
2.4.2:
2.5:

Figure E.3: A sequence diagram describing the process for checking for fulfilled trigger conditions at two different times. There is only one active trigger condition in the server. The loop located in kwpm_pwpc.c runs every 10ms and calls the function do_roe() which checks all the active trigger conditions. In the first check, the trigger condition is not fulfilled, but in the second it is. To execute and create the response message according to the serviceToRespondTo, roe_tms.c call ckwp_kwps.h to get a function pointer to the diagnostic service defined by the serviceToRespondTo. The response generated by calling the function pointed to is then sent on the CAN bus by using the function Coms_putCanTxMsg_E in coms.h.

F C300 UML Diagrams

This appendix contains UML diagrams describing the software architecture in the C300 that is of importance to the client application. This appendix addresses the interested reader who have a basic understanding of Scania's software architecture.

F.1 System description

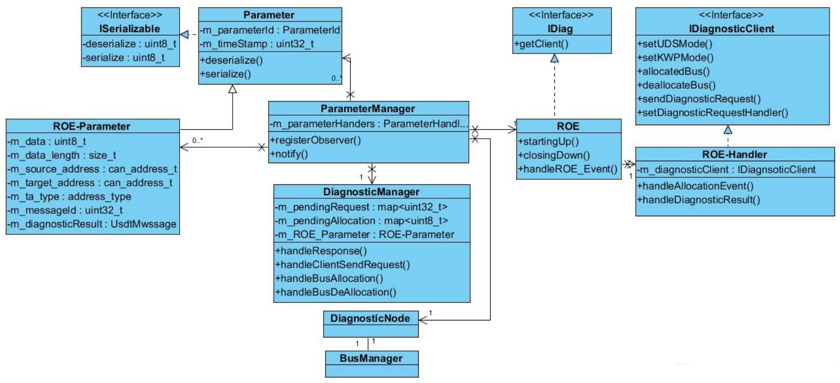

Figure F.1: System description of the prototype. ROE contains the client application and ROE-Handler handles the diagnostic requests and responses. IDiagnosticClient is a interface to send and receive diagnostic messages. ROE-Parameter contains information related to the event message and is used to pass information between the classes.

F.2 Sequence diagram of request message

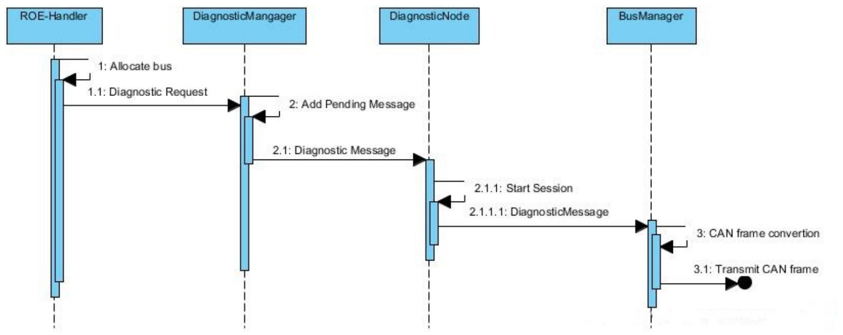

Figure F.2: Sequence diagram describing the process for sending a diagnostic request message. ROE-Handler: Creates a request message and calls DiagnosticManager to put the message in the outgoing message queue. The DiagnosticNode sets up a session and starts a timer for the communication. The BusManager is told to transmit the message on CAN.

F.3 Sequence diagram of response message

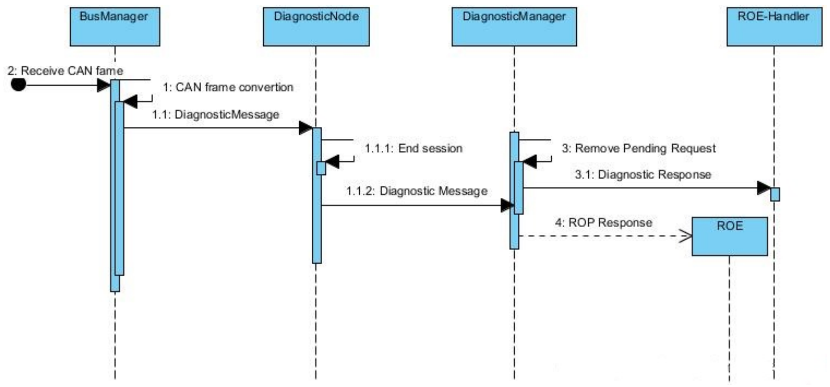

Figure F.3: Sequence diagram describing the process for receiving a response message. The BusManager receives the message and creates a DiagnosicMessage out of it. DiasgnosticNode closes the session and DiagnosticManager removes the corresponding request from its pending request queue if there are any and calls ROE-Handler. If there is no pending request in the queue, DiagnosticManager assumes it is an event generated message and it then calls ROE.

Bibliography

[1] Philip Burnard. “A method of analysing interview transcripts in qualitative research”. In: *Nurse education today* 11.6 (1991), pp. 461–466.

[2] ISO. *Road vehicles – Diagnostic systems – Keyword Protocol 2000 – Part 3: Application layer: ISO 14230-3*. [Scania provided document]. 1999.

[3] Sanna Talja. “Analyzing qualitative interview data: The discourse analytic method”. In: *Library & information science research* 21.4 (1999), pp. 459–477.

[4] ISO. *Swedish Implementation Standard - Road Vehicles – Diagnostic systems – Keyword Protocol 2000 – Part 3: Application layer: ISO 14230-3*. [Online accessed 8 February 2016 url: http://www.alfa145.co.uk/obd/14230-3s.pdf]. 2001.

[5] Steve Corrigan. *Introduction to the Controller Area Network (CAN)*. Tech. rep. Revised 2008. Texas Instruments, 2002.

[6] Lauesen Soren. *Software Requirements, Styles and Techniques*. Pearson Education Limited, 2002.

[7] Karimi Ali, Olsson Johan, and Rydell Johan. “A Software Architecture Approach to Remote Vehicle Diagnostics”. MA thesis. Göteborg: Chalmers, 2004.

[8] Ulla H Graneheim and Berit Lundman. “Qualitative content analysis in nursing research: concepts, procedures and measures to achieve trustworthiness”. In: *Nurse education today* 24.2 (2004), pp. 105–112.

[9] Axelsson Jakob et al. *The Industrial Information Technology Handbook, chapter 57: Vehicle Functional Domains and Their Requirements*. CRC Press, 2004. ISBN: 978-1-4200-3633-6.

[10] National Instruments. *Controller Area Network (CAN) Overview*. [Online; accessed 14-February-2016 url: http://www.ni.com/white-paper/2732/en/toc5]. 2004-08-01.

[11] Marscholik Christoph and Subke Peter. *Road Vehicles Diagnostic Communication*. Hüthig, 2005.

[12] Siw Elisabeth Hove and Bente Anda. “Experiences from conducting semi-structured interviews in empirical software engineering research”. In: *Software metrics, 2005. 11th ieee international symposium*. IEEE. 2005, 10–pp.

[13] Nicolas Navet and Françoise Simonot-Lion. *Automotive embedded systems handbook, chapter 4: A Review of Embedded Automotive Protocols*. CRC press, 2008.

[14] Marco Di Natale. *Understanding and using the Control Area Network*. [UML: http://inst.cs.berkeley.edu/ ee249/fa08/Lectures/handout$_{c}anbus2.pdf.Accessed2016 - 05 - 08$]. 2008-10-30.

[15] Marco Di Natale. *Understanding and using the Controller Area Network*. [Online accessed: 3 April 2016. Url: https://inst.eecs.berkeley.edu/ ee249/fa08/Lectures/handout$_{c}anbus2.pdf$]. 2008-10-30.

[16] Martyn Denscombe. *Forskningshandboken: för småskaliga forskningsprojekt inom samhällsvetenskaperna*. Studentlitteratur, 2009.

[17] M. Galla Thomas. *Network Embedded Systems: Standardized System Software for Automotive Applications*. CRC Press, 2009.

[18] Richard Zurawski. *Network Embedded Systems: An Introduction. Chapter 1, An Overview*. CRC Press, 2009.

[19] Graham Pitcher. *Growing number of ecus forces new approach to cars electrical architecture*. Sept. 2012. URL: `http://www.newelectronics.co.uk/electronics-technology/growing-number-of-ecus-forces-new-approach-to-car-electrical-architecture/45039/` (visited on 05/02/2016).

[20] Erik Rautalinko. “Reflective listening and open-ended questions in counselling: Preferences moderated by social skills and cognitive ability”. In: *Counselling and Psychotherapy Research* 13.1 (2013), pp. 24–31.

[21] Swedish Implementation Standard. *Road vehicles – Unified diagnostic services (UDS), Part 2: Session layer services, ISO 14229-2:2013*. [Scania provided document]. 2013.

[22] International Organization for Standardization. *Road vehicles – Unified diagnostic services (UDS), Part 1: Specification and requirements, ISO 14229-1*. 2013.

[23] Shanwell Truls and Svensson Håkan. “Remote diagnostics of heavy trucks through telematics”. MA thesis. SE-100 44 STOCKHOLM: KTH, 2013.

[24] Wohlin Claes. *Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering*. 2014. URL: `http://www.wohlin.eu/ease14.pdf` (visited on 05/04/2016).

[25] *CAN Bus and OBD 2, with Examples of how CAN Bus and OBD 2 Work!* 2014-02-06. URL: `http://www.oneminuteinfo.com/2014/02/can-bus-obd-ii-explained-for-dummies.html` (visited on 05/08/2016).

[26] SAE. *J1939$_{2}$1 : 201504*. 2015.

[27] Scania. *Technical product data, System description TMS2, doc.nr: 2335499*. 2015-08-10.

[28] Scania. *Technical product data, System description C300, doc.nr: 2183519*. 2015-12-11.

[29] Scania AB. *Company overview*. [Online; accessed 02-February-2016]. 2016. URL: `http://www.scania.se/om-scania/scaniakoncernen/`.

[30] Kvaser. *CAN Protocol Tutorial. Introduction: The CAN bus @ONLINE*. Apr. 2016. URL: `https://www.kvaser.com/can-protocol-tutorial/`.

[31] Kvaser. *J1939 Introduction @ONLINE*. Apr. 2016. URL: `https://www.kvaser.com/about-can/higher-layer-protocols/j1939-introduction/`.

[32] RESD. *PowerPoint Presentation by Andreas Jonasson (RESD)*. 2016.

[33] B. *Development Engineer - Driver Assistance Controls*. 2016-01-02.

[34] C. *Senior Engineer - Embedded Tools and Test Environment*. 2016-01-04.

[35] A. *Senior Engineer - System and Integration Test*. 2016-01-22.

[36] Scania. *Read out of the number of ECUs in one of Scania's test vehicles.* 2016-02-02.

[37] B. *Senior Engineer - Diagnostic Architect*. 2016-02-05.

[38] United States Environmental Protection Agency. *On-Board Diagnostics*. 2016-02-24. URL: `https://www3.epa.gov/obd/basic.htm` (visited on 05/03/2016).

[39] E. *Software Developer, Remote Diagnostics*. 2016-02-25.

[40] G F. *Senior Engineer - Driver Assistance Controls, Development Engineer - Driver Assistance Controls*. 2016-02-25.

[41] Andreas Carlén. *Development engineer*. 2016-03-09.

[42] Anna Beckman. *Technical Manager, Electrical System Safety*. 2016-03-30.

[43] Annika Westrand. *Software Design Engineer*. 2016-04-19.

[44] Eskilson Anders. *Software Design Engineer*. 2016-05-15.

[45] Hans Törnquist. *Development engineer*. 2016-05-15.

[46] Eskilson Anders. *Software Design Engineer*. 2016-05-23.

[47] Volvo. *Recycling Manual L60F, L70F, L90F, L110F, L120F*. URL: `https://www.volvoce.com/SiteCollectionDocuments/VCE/Documents%20Global/VCE%20Corporate/Recycling_Manual_L60F___L120F.pdf` (visited on 05/02/2016).

EXAMENSARBETE Event based diagnostics on heavy duty vehicles
STUDENT Johan Winér
HANDLEDARE Sven Gestegård Robertz (Scania)
EXAMINATOR Björn Regnell (LTH)

Eventtjänst för diagnos av tunga fordon

POPULÄRVETENSKAPLIG SAMMANFATTNING **Johan Winér**

För att kunna diagnosera allt mer avancerade funktionerna i fordon måste mer data läsas ut ur fordonets inbyggda datorenheter. Men nu är gränsen nådd för vad fordonsnätverket klarar av. Därför har en eventtjänst undersökts som ett sätt att minska nätverkslasten.

Under åren har antalet datorenheter i fordon ökat och med dem kommer också möjligheten till mer avancerade funktioner. För att felsöka dessa behövs mer data läsas ut ur enheterna vilket lett till en ökad last på fordonets nätverk. För att möjliggöra diagnoseringen av de mer avancerade funktionerna måste nätverkslasten som diagnosfunktionerna ger upphov till minskas. Därför tittar man nu efter nya sätt att skicka data mellan enheterna. Ett alternativ är en eventtjänst.

Vid diagnos av fordon läser man idag datan som man är intresserad av med jämna intervall, så kallad sampling. Det finns dock ingen garanti för att datan ska ha ändrat sig sedan förra gången den lästes vilket kan leda till onödig trafik på nätverket om man bara är intresserad av att upptäcka ändringar. M.h.a. en eventtjänst kan man istället för sampling, låta enheten som har datan själv ta ansvar för att skicka den när en viss händelse (event) har inträffat. Detta kan ha stora positiva effekter på nätverkslasten. En eventtjänst har därför undersökts med syfte att utreda hur behovet av en sådan tjänst ser ut och hur den skulle implementeras.

Det finns redan en beskrivning av hur en eventtjänst för diagnos av fordon skulle kunna se ut. Beskrivningen står i en standard och beskriver en tjänst som kallas ResponseOnEvent (ROE). Denna tjänst undersöktes för att se vilka problem som finns med en eventtjänst och hur en eventtjänst skulle se ut för att kunna möta behoven från olika potentiella användningsområden.

När ROE jämfördes med de funna områdenas behov visade det sig att en del av behoven inte skulle gå att stödja med ROE. Därför togs en alternativ design fram som byggde vidare på ROE men också gjorde vissa begränsningar för att lösa vissa problem som hade upptäckts.

Ett av de största problemen som upptäcktes var att det inte fanns stöd för att låta en datorenhet själv skicka data utan att den först fått en förfrågan om att skicka datan. Kommunikationen av data hade alltid skett genom att en förfrågan skickats till enheten som den sedan svarat på. För att kunna implementera tjänsten skulle en omarbetning av mjukvaran i datorenheterna behöva göras vilket skulle kunna kräva några månaders arbete. Ett annat problem som också upptäcktes var att tjänsten skulle komma att kräva mer minne än vad man först räknat med. Då mängden minne i datorenheterna är mycket begränsad så skulle detta komma att påverka användbarheten av tjänsten. Skulle tjänsten komma att ta upp för mycket minne skulle den eventuellt inte gå att implementera alls i vissa enheter. Mycket av designvalen kring den alternativa tjänsten var därför en övervägning mellan flexibilitet och storleken av tjänsten.

Slutsatserna av arbetet är att det finns begränsningar i tjänsten ROE som gör det fördelaktigt att istället ta fram en egendesignad tjänst anpassad för de behov man ser. En eventtjänst har stor potential att minska nätverkslasten men den kan eventuellt inte göras så flexibel som man hade hoppats p.g.a. den begränsade mängden minne i datorenheterna. För att kunna implementera tjänsten kan det också krävas en del omarbetningar av den befintliga mjukvaran för att stödja eventbaserad kommunikation.

De upptäckta begränsningarna och problemen med en eventtjänst kommer kunna underlätta en framtida design och implementation av en eventtjänst och de lösningsförslag som undersöktes kan förhoppningsvis användas med fördel.