

Mobile Informatics

PowerTools

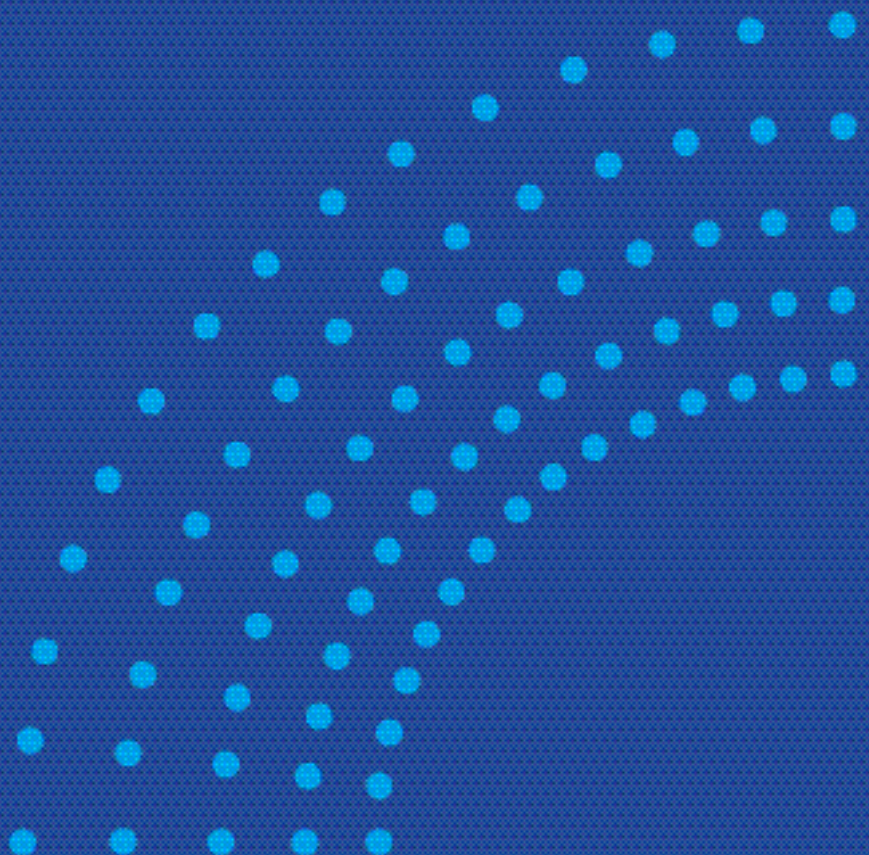
Designing for mobility with web based diagnostics for Volvo Cars

Kaveh Azimzadeh

Göteborg, Sweden 2002



IT University
of Göteborg
CHALMERS | GÖTEBORGS UNIVERSITET



REPORT NO. 2002:7

PowerTools

Designing for mobility with web based diagnostics for Volvo Cars

Kaveh Azimzadeh



Department of Applied Information Technology
IT UNIVERSITY OF GÖTEBORG
GÖTEBORG UNIVERSITY AND CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2002

PowerTools

Designing for mobility with web based diagnostics for Volvo Cars

Kaveh Azimzadeh

© KAVEH AZIMZADEH, 2002.

Report no 2002:7

ISSN: 1651-4769

Department of Applied Information Technology

IT University of Göteborg

Göteborg University and Chalmers University of Technology

P O Box 8718

SE – 402 75 Göteborg

Sweden

Telephone + 46 (0)31-772 4895

[tryckeriets namn]

Göteborg, Sweden 2002

PowerTools

Designing for mobility with web based diagnostics for Volvo Cars

KAVEH AZIMZADEH

Department of Applied Information Technology

IT University of Göteborg

Göteborg University and Chalmers University of Technology

SUMMARY

This is a report for a Master Thesis project in Mobile Informatics carried out at Volvo Cars Costumer Service in Gothenburg, Sweden. This master thesis project explore the use of Internet technologies within the research areas Mobile Informatics and Telematics. The project was carried out as a part of an ongoing project, Volvo Aftersales and Diagnostics Information System Next Generation (VADIS NG) at Volvo Cars Costumer Service (VCCS).

The project and the research within were defined by Volvo Cars as follows: *Develop a vehicle communication system using existing vehicle communication interface. The system shall be based on such architectural principals so that it can be deployed in various infrastructure scenarios predefined for the new version of VADIS/VADIS2004.* The outline of this master thesis project was set by VADIS NG, which is an ongoing project at Volvo for developing a new version of the existing workshop tool system, VADIS.

To meet the requirements defined, a layered, system architecture was presented. The system introduced the use of .NET based web services and rich applications, PowerTools, developed in Macromedia FLASH and new XML-based Scripts, PowerScript for definition and control of the PowerTools applications. This layered architecture allows the system to be configured differently based on user mobility, user preferences and user task.

A functional prototype was developed for demonstration of the system. The application was developed in Macromedia FLASH using .NET web services developed in C#. The prototype demonstrated simple diagnostics sessions using PowerTools for monitoring and controlling various features and functions in the vehicle, for example monitoring the vehicle engine speed and running condition or turning on the headlights.

The technical and architectural principals presented within the project would be the same as principals used for a Telematics system. Hence a future implementation of such services for third party service suppliers can be based on the same ideas.

The system and the prototype were evaluated at a reference group meeting. The mobile aspects of the system architecture were demonstrated by remotely controlling the vehicle. The results reached by this Master Thesis Project will hopefully help the development of the Vehicle Communication parts of VADIS2004 and function as a technical reference base for future implementation and deployment of the system in various infrastructure scenarios

Keywords: Mobility, Vehicle Diagnostics, OBD, Web Services, Flash, Layered Architecture

Acknowledgments

This master thesis project has been carried out at Volvo Cars in Gothenburg, Sweden. It has been a great opportunity for me to be involved in an interesting project and thereby gain skills and knowledge in the area. I would like to take the opportunity to thank all the people, specially the members of the system group in the VADIS NG project that have given me help and support during the work.

I would further more like to greatly thank my supervisor, Daniel Olsson for his insightful guidance and continuous assistance through out the project.

Gothenburg 2002.12.15

Kaveh Azimzadeh

PowerTools

*Designing for mobility with web based
diagnostics for Volvo Cars*



Index

1	INTRODUCTION.....	6
1.1	READER'S GUIDE.....	6
1.2	MASTER THESIS OBJECTIVE.....	6
2	BACKGROUND.....	7
2.1	VOLVO CARS CUSTOMER SERVICE AND VADIS.....	7
2.2	VADIS NEXT GENERATION - VADIS2004.....	9
2.3	DIAGNOSTICS.....	10
3	MOBILE INFORMATICS AND TELEMATICS.....	12
3.1	MOBILE INFORMATICS.....	13
3.2	TELEMATICS.....	14
3.3	TELEMATICS AND THIS PROJECT.....	15
4	POWERTOOLS.....	16
4.1	PURPOSE OF THE PROTOTYPE.....	16
4.2	PROBLEM AND SOLUTION.....	17
4.3	POWERTOOLS AND MOBILITY ISSUES.....	18
5	DESIGNING THE SYSTEM.....	19
5.1	METHODOLOGY.....	19
5.2	DELIMITATIONS.....	23
6	DEVELOPMENT.....	25
6.1	WEB SERVICES.....	25
6.2	LAYERED ARCHITECTURE.....	26
6.3	PROGRAMMING IN MICROSOFT .NET.....	27
6.4	MACROMEDIA FLASH.....	29
6.5	PROBLEMS AND ISSUES.....	32
7	EVALUATION / VALIDATION.....	33
7.1	EVALUATION.....	33
7.2	VALIDATION.....	34
8	SUMMARY.....	36
8.1	OBJECTIVES AND ACHIEVED RESULTS.....	36
	ACRONYMS.....	37
	REFERENCES.....	38
	APPENDIX A.....	40
	TECHNICAL SPECIFICATIONS.....	41
	SYSTEM ARCHITECTURE.....	41
	CLASSES AND METHODS.....	42
	POWERSCRIPT / POWERTOOLS.....	47
	PROTOTYPE APPLICATION.....	49
	APPENDIX B.....	51
	ELECTRONIC COMPUTER UNIT CONTEXT MODEL.....	52
	APPENDIX C.....	54
	TELEMATICS.....	55

1 Introduction

This report is the final report for a Master Thesis Project at Volvo Cars Co. The project will result in a Masters Degree at Mobile Informatics at IT-university in Gothenburg Sweden.

1.1 Reader's guide

This report is divided in three major parts; the first part (chapter 1 - 3) contains information for the reader to be able to understand the carried out project on a deeper level. This is done by giving some background information on the department at Volvo where the project was carried out and related existing systems at Volvo Cars Co and also by presenting the area of Mobile Informatics and Telematics and the concept of this project within the frame of these fields.

The second part (chapter 4 - 7) contains information regarding the methodology and the technologies used to design and develop the system along with a presentation of the PowerTools concept.

The technical specifications for the designed system and the prototype application can be found in Appendix A. Here the reader will find detailed information on each part of the system.

One of the main goals of this master thesis report in particular was to function as a technical reference base for future implementation of the ideas presented. Therefore parts of it can be of a technical character.

To ease the understanding of the report there is an Abbreviation list at the end of the report.

1.2 Master Thesis Objective

The main objective of this project is to build a technical reference base for future development and implementation. This project will result in a functional prototype. The prototype will be included in the first release of VADIS2004 (V0.1). The idea is to present a system design that can meet the requirements set by the architectural principals for the main system (VADIS2004).

The purpose of this project can be summarized as follows:

1. Present an attractive system design including a collection of .NET web services for car communication based on *iVIC dll:s*
2. Build a functional prototype that can be integrated within the VADIS2004 framework. The prototype will present PowerTools (see chapter 4) functionality supported by the system.
3. Highlight problems and propose technical solutions within the problem area
4. Documentation that will present the carried out project and the ideas presented, in written form.

2 Background

This master thesis project will explore different areas within Mobile Informatics and Telematics and the use of Internet technologies within the research area.

The goal of this project was set to present a functional prototype, demonstrating the communication to a car through visual diagnostic applications called, PowerTools. The functionality and the performance logic for these tools were to be defined by special designed scripts written in XML [23] (PowerScript). This way the system architecture would be designed as a layered system with XML interface between the layers.

This project has been carried out at Volvo Cars Corporation Customer Service in Gothenburg as a part of an ongoing project, Volvo Aftersales and Diagnostics Information System Next Generation (VADIS NG).

2.1 Volvo Cars Customer Service and VADIS

2.1.1 Volvo Cars Customer Service

The VCCS department at Volvo is responsible for Volvo Car Corporation aftermarket business. During the life cycle of a Volvo car Customer Service has major functions. These functions are presented in Figure 1 below.

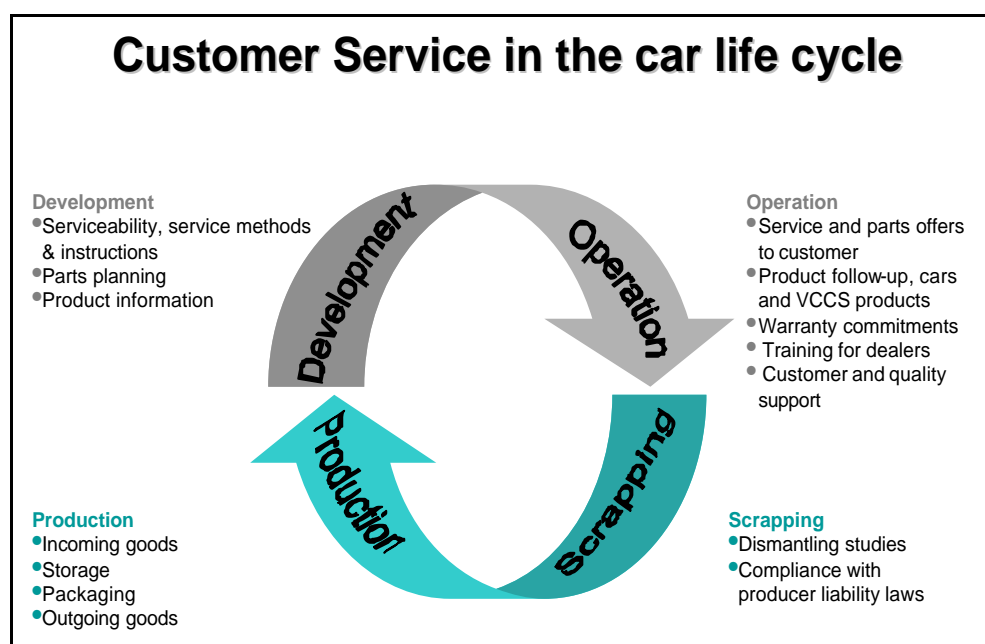


Figure 1 – Customer Service in the car life cycle

The VADIS NG project at Volvo Cars Customer will result in a system that will cover both the development and operation stages of the car life cycle as shown in Figure 1.

2.1.2 VADIS

VADIS – Volvo After sales and Diagnostics Information System – is a PC based Information System designed for VCC Workshops that was put into the market in 1995 and has been refined since then. VADIS can be used to find parts information and service information. It can handle car diagnostics and software download.

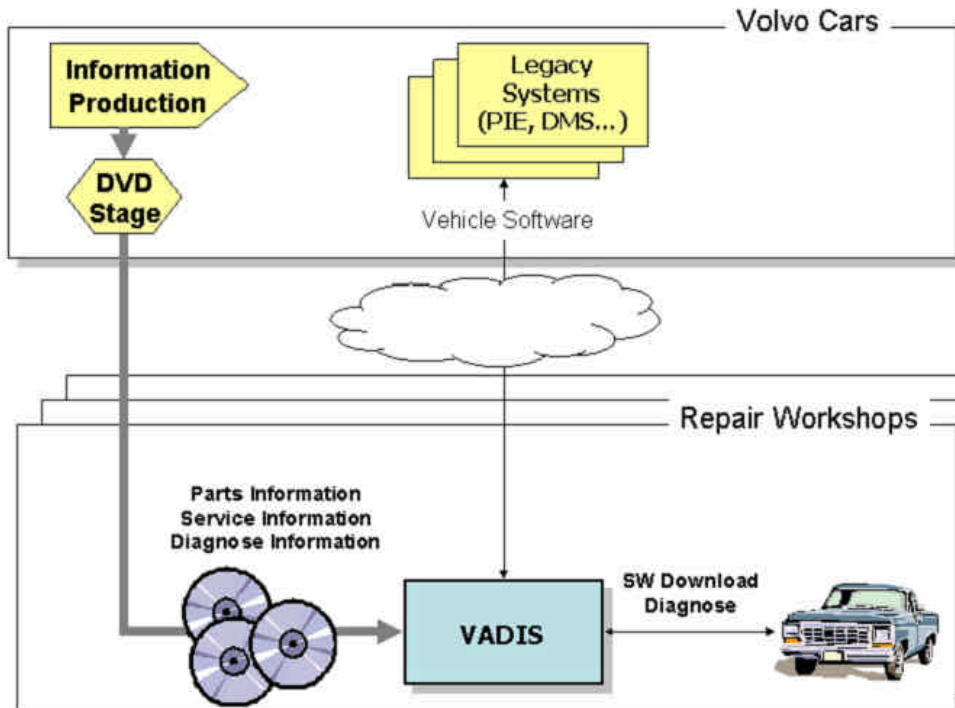


Figure 2 – The VADIS System is used at repair shops globally. They are feed by information produced at Volvo and they can fetch vehicle software from legacy systems at Volvo.

VADIS has a user interface, a vehicle interface, and interfaces to other systems. The User Interface comprises a monitor (display), keyboard and pointer (mouse and Touch screen). VADIS is today installed on either an InfoPC or on a VADIS Cart.

The main purpose for the VADIS system today is to support the authorized dealership in repairing and servicing the cars. VADIS provides service and parts information, as well as diagnostic fault tracing and software download. Some properties of VADIS:

- Thick client: VADIS application software on each VADIS cart
- Mostly offline, sometimes online
- Manual updates
- DVD distribution
- Robust solution
- Difficult/impossible to customize for different users



Figure 3 – The VADIS cart. it consist of a standard PC wrapped in a metal case. It is equipped with touch screen, mouse

2.2 VADIS Next Generation - VADIS2004

The Project VADIS Next Generation will result in a system called VADIS2004 that will replace VADIS in 2004. VADIS2004 comprises most of the services that are available today, as well as new services. The Project will create a new business concept for the VADIS2004 as well as develop and introduce the VADIS2004 to the market. Among the purposes of the new design solution are to reduce VADIS system change lead-time, and increase customer satisfaction. Some highlights of VADIS2004:

- Online system
- Thin client, typically only a browser
- Offline possibilities with limited functionality
- Simplified distribution, Information-updates via web, via DVD possible in special situations
- Flexible and scalable solution,
- Enable mobility
- Easy and generic deployment of the system,
- Improved helpdesk,
- Global availability, support volume growth,
- Incremental update, user feedback

VADIS2004 will have a component-based solution. This will decrease change-, and development-lead-time, thus cutting cost for the organization and enabling the development of the VCCS business.

The VADIS NG project is an ongoing project at Volvo Cars Customer Service and this master thesis project is part of the system development part of the project. The system designed within this master thesis project is based on the same architectural principals as the VADIS NG system. The prototype system developed within this project will be a part of the first release of VADIS 2004 V.01, demonstrating the vehicle communication parts of the system. The logical system units of the VADIS 2004 system are illustrated below in Figure 4.

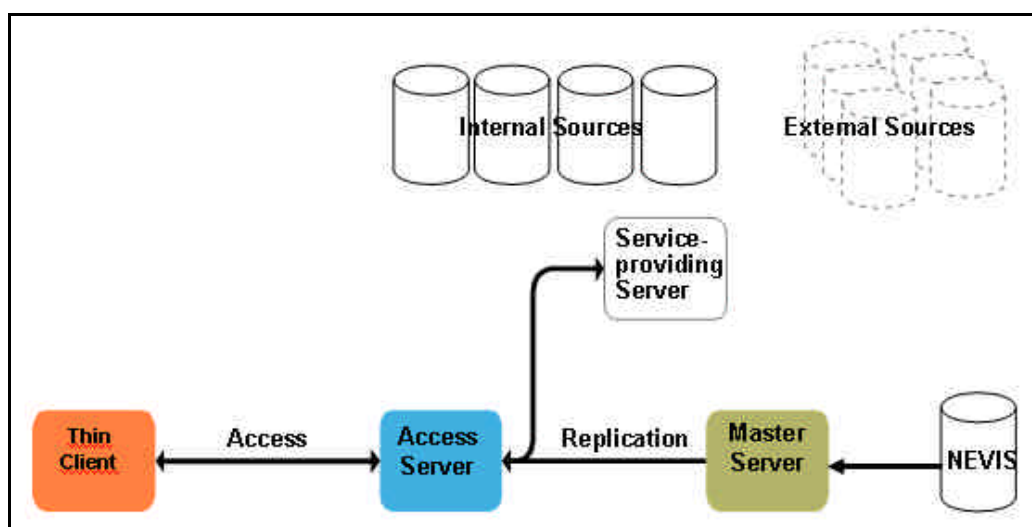


Figure 4 – Logical system unit of VADIS2004

2.3 Diagnostics

Diagnostics on cars was a simple concept back in the early stages of the automobile history in the beginning of the 19th century. Back then, the mechanics only had to handle the common mechanical issues when tracing faults and performing diagnostics on the vehicle. Well today, the fault tracing and diagnostics process on cars has totally changed and is based on other factors. A modern automobile of today contains numerous electric controllers which operate with advanced software. Almost all functionality in the car from the adjustment of the heater/AC system to the more advanced engine control systems is controlled by special software.

Obviously today's mechanics need more advanced tools for diagnostics of the cars. Tools that can communicate with the advanced and complex control unit system and the software's that control it.

VADIS, previously described in this chapter, is an example of such tool used in Volvo Cars repair shops and production units. When developing software tools of VADIS kind developers must have deep understanding of the specific vehicle communication concepts.

2.3.1 OBDII

OBD-II is a relatively new standard introduced in the mid-'90s. It provides almost complete engine control and also monitors parts of the chassis, body and accessory devices, as well as the diagnostic control network of the car [2].

Initially there were few standards and each manufacturer had their own systems and signals. In 1988, the Society of Automotive Engineers (SAE) [20] set a standard connector plug and set of diagnostic test signals. The Environmental Protection Agency (EPA) [7] adapted most of their standards from the SAE on-board diagnostic programs and recommendations. OBD-II is an expanded set of standards and practices developed by SAE and adopted by the EPA and CARB (California Air Resources Board) [3] for implementation by January 1, 1996.

Volvo cars use a Volvo specific variant of the OBDII protocol, called Volvo Diagnostics II.

2.3.2 Volvo On-board Diagnostic

The Volvo Diagnostic II concept specifies requirements for on-vehicle Electronic Control Units (ECU) such that a tester (see Appendix B) system can control diagnostic functions on the ECU through a serial link. The concept is based on the ISO [14] and SAE diagnostic standards with adaptation to specific Volvo and legal requirements.

The physical location of the various electronic control modules (ECU nodes) in the car and their interconnection with two different multiplex networks is shown in Figure 5. The CEM module provides a 'bridge' between the two networks, enabling information to be exchanged between them. As already noted, many functions are divided between several electronic modules. The alarm function, for example, is split between the CEM and 5 other modules.

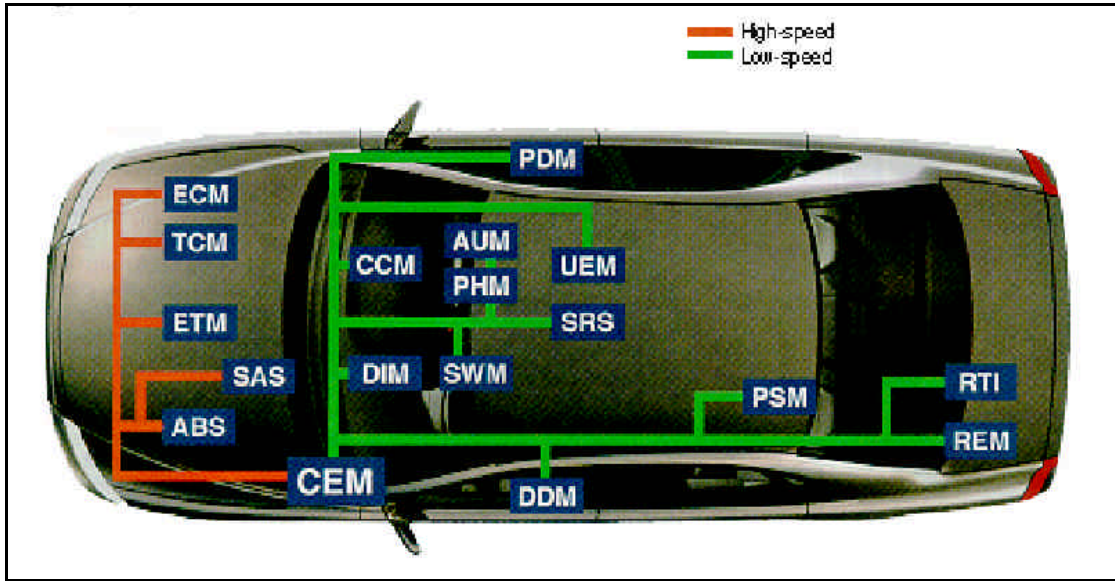


Figure 5 – Electrical Computer Unit node map

The prototype developed within this master thesis project illustrates control of diagnostic functions on certain chosen ECU nodes. The context model of ECU and its communication interface is further described in Appendix C. The prototype application specifications can be found in Appendix A.

3 Mobile Informatics and Telematics

This section provides a brief outline of Mobile Informatics and Telematics and relates them to the research communities IEEE [12] and ACM [1]. This is then used as backdrop when pinning down the area where this project resides.

Mobile Informatics is an applied research field concerned with the use of new applications for mobile settings. The objective is to explore, design and evaluate innovative ways of using Information Technology [13].

Adapting the use of modern IT with a design objective for supporting mobility is characterizing for R&D within Mobile Informatics [5]. Such an approach for system design is usually the case for Telematics systems.

Telematics covers the field of IT-services provided for modern vehicles using different technologies. The theoretical approach for a Telematics project, when it comes to methodology, design and development is substantially the same as projects within Mobile Informatics [21]. Hence, Telematics can be categorized as a block within the Mobile Informatics field.

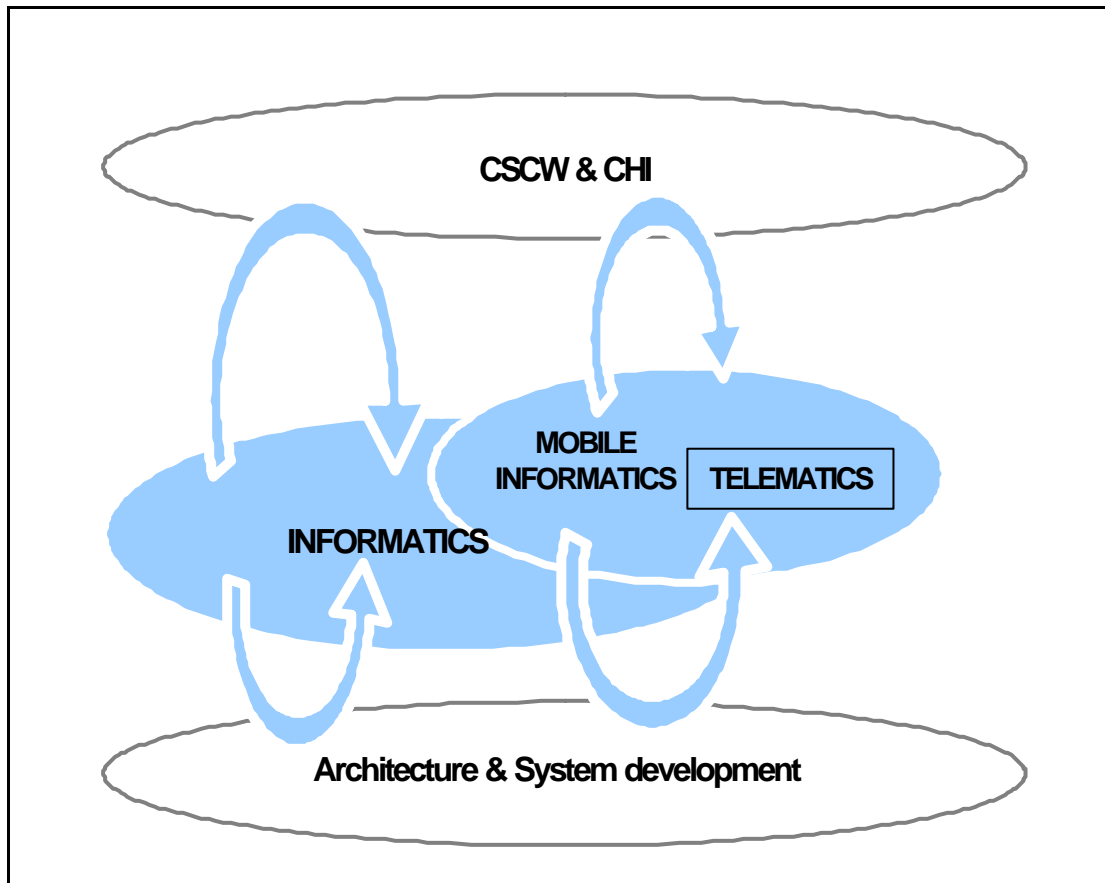


Figure 6 - The “big picture” concept

Figure 6 above is an attempt to describe how the mentioned research areas stand in relation to each other in the “big picture”.

The CSCW (Computer Supported Cooperative Work) and CHI (Computer Human Interaction) communities work with R&D, furthermore there is a focus on issues such as social

and environmental (in the sense of context) aspects of computerization. These communities are linked to the ACM society. The applied research areas of Informatics, Mobile Informatics and Telematics are in many aspects influenced and inspired by the CSCW and HCI theories. For instance in Mobile Informatics methodology, ethnographic field studies which is taken from the CSCW world is used with an ultimate design ambition. However, the Mobile Informatics field along with Telematics does also take advantage of the applied theories of modern architecture and system development. These disciplines are addressed in the CSCW/CHI communities, but not as thorough as in the IEEE community [12], which is an organization with focus on system design and development.

3.1 Mobile Informatics

Mobile Informatics is an applied research area exploring mobile concepts and mobile information technology. The research is design oriented. Traditionally Mobile Informatics combines field studies with technological innovations and research knowledge. The research is often business oriented and results in innovative concepts and knowledge for the research community and also attractive solutions for the industry.

According to Bo Dahlbom et al. [5] we are initially faced with a problem and a subject, we approach the subject when laying the structure base for a system design, but instead of going on about developing an Information System, we are to define our discipline in terms of using Information Technology, and when that use is focusing more on mobile IT support, we are in the field of Mobile Informatics.

To be able to relate to mobile IT support for D&D (Design and Development) of the system, we need to clarify the existing mobility aspects in the project context. For starters *mobility* must be defined, this is however not an easy task [8]. A point of view is to define mobility based on the modality, which is the physical relocation of the *mobile* subject [15]. Kristoffersen and Ljungberg present the following model for the so called “mobile work (work, in the sense of an activity)”.

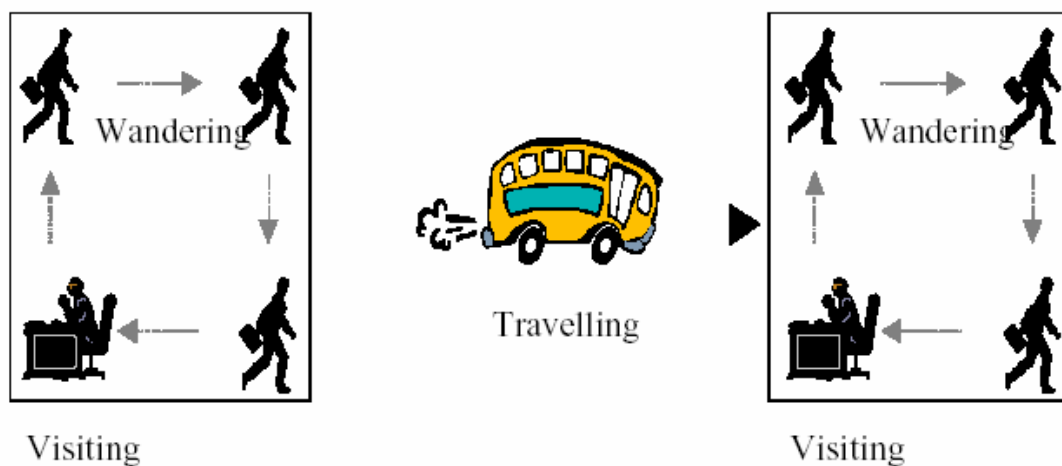


Figure 7 – Modalities of mobile work [15]

Based on this model we can categorize the mobility in a vehicle repair workshop as a “visiting” modality where the mobile users “wander”. In the Workshop environment the type of mobility is mainly related to the physical movement of the mechanics (the worker or the user), and an ethnographic study can identify a pattern for the mobility and thereby design

implications for supporting services can be made. This was also the case for this master thesis project.

However, there is another more complex type of mobility in the context of modern vehicle diagnostics. This is when we are talking about so called remote diagnostics or automated live diagnostics. Meaning that a diagnostic session takes place during the period the vehicle is in on the move.

In a case where IT-provided services are meant for the passengers of a moving vehicle, the passengers become the “mobile users” in a “Traveling” modality and the car is just the means for transportation.

In the case of mobility in the workshop, the mobile subject was the mechanic and not the vehicle, therefore the design implications didn't need to take the vehicle itself in consideration. But in the case where the car is in movement on the road and is being remotely diagnosed, the subject of mobility is the vehicle itself. Thus the vehicle can be considered as the target for the provided service. Here, there is a completely different design scenario. The design implications for instance, are based on other technical factors than the workshop scenario. This case imposes bigger demands on an infrastructure supporting the services.

The system developed within this project was designed with consideration to the mentioned mobility issues. The system is for example designed in such way that the client/user side machine can and most likely will be a mobile unit supporting the mobile work of the mechanic in the workshop environment. The use of a so called “Tablet PC” [19] is supported by the system. Also smaller mobile devices such as Compaq iPAQ running Pocket PC can be supported through minor modifications regarding the presentation view of the system application.

The system built also supports remote diagnostics as mentioned earlier, which means that a vehicle on the road can be supported and diagnosed remotely through an internet web-site. Remote diagnostics does in some sense result in possible remote working, meaning that a mechanic who till now had to consider the workplace exact same physical location as the car now has the possibility to work at distance, hence becoming a remote worker [6].

Support for remote on-road diagnostics is a pure Mobile Informatics solution, but is more suited under the new term of “Telematics” which is described in the next session in this chapter.

The mobility issues related to the system architecture are further discussed in chapter 5.1.3

3.2 Telematics

Telematics is a relatively new concept within Mobile Informatics. The term Telematics roughly described is the definition term for all computer based vehicle services using different telecommunication technology. The focus is today mainly on wireless and positioning services. Although many Telematics projects have been carried out at major companies around the world, there is still very limited amount of academic research done in the area.

Telematics Services rely on an infrastructure to function. Such an infrastructure exists today but is continuously evolving. Telematics is further described in detail in Appendix C.

3.3 Telematics and this project

The Telematics market is still in its initial faze and is changing continuously. This considers all four major actors in the market. Because of the changes and the variation of the type and content provided by the Telematics market, it is of major importance that the OEMs (Original Equipment Manufacturers) in this case Volvo Cars build the systems in such way that they can support changes.

Volvo is like most other manufacturers of premium cars introducing hardware and software support in their new car models for use and implementation of future Telematics services. An example of this is the latest Volvo SUV (Sports Utility Vehicle) vehicle model that has recently been introduced by Volvo Cars. This car uses, except the ordinary CAN (Car Area Network) – network a so called MOST (Media Oriented Systems Transport) network for communication with multimedia devices in the vehicle such as DVD players.

The system deigned within this project does present the use of specific diagnostic Telematics services. But the architecture of the system is designed in such way that it can also be adapted for use of other Telematics services. This has been one of the goals for this project.

The system is designed with a layered structure. By splitting the system into layers we gain flexibility. This is important when it comes to implementation of Telematics services from external service- and content providers. A layered structure makes it possible to by need, only replace the part that is in charge for the special task instead of creating an entirely new system. Designing the system with a layered structure is therefore very cost effective when new features and external system dependencies must be implied.

Furthermore the system architecture presented in this project allows for retrieval and remote monitoring of vehicular diagnostic information. Preemptive monitoring of vehicular diagnostics is an interesting Telematics category that offers significant advantages to both the consumer and the vehicle manufacturer. Automated services for remote detection and fault fix can improve consumer safety and reduce incidence of vehicle related problems, consequently reducing repair and part replacement expenses for the manufactures during the warranty period.

4 PowerTools

PowerTools (Introduced within this project) is the chosen name of the diagnostic applications for advanced vehicle communication in the coming VADIS2004 System. A number of PowerTools along with a main control application has been developed in this project as prototypes to illustrate the core development and functionality theories of the idea. The PowerTools applications are part of a bigger system, a web based system with a layered architectural structure. The system model is illustrated in Figure 8.

The next sections in this chapter explain the purpose of PowerTools applications in general and the background reasons for development of such applications. A more detailed and technical presentation of PowerTools applications can be found in Appendix A along with full description of the entire system.

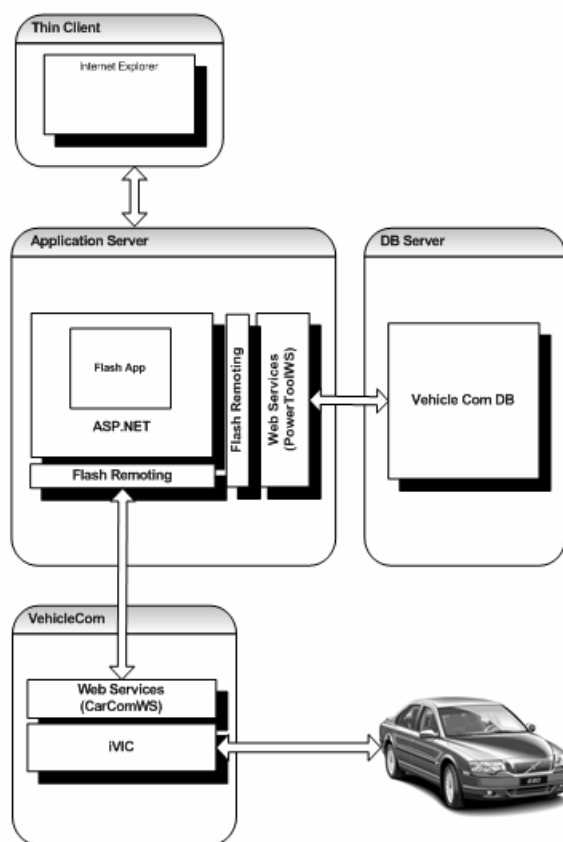


Figure 8 – PowerTools system model

4.1 Purpose of the prototype

To be able to perform diagnostics on modern Volvo vehicles that are equipped with system of Electrical Control Units (ECUs), there is a need for software applications that can interact with that system. The so called PowerTools applications are needed for advanced vehicle communication where different types of control and check actions are performed on the present ECUs. For instance a diagnostic session where the vehicle engine parameter values are to be checked can be performed through a PowerTool that communicates with the certain ECU controlling the vehicle engine.

However, what just described is no more different than the functionalities that exist in the current version of VADIS system. What differs is the fact that PowerTools are intended to be part of a totally different system with different architecture, therefore the requirements are different. The objective is consequently to develop diagnostic vehicle communication applications (Prototypes) based on, in this project suggested architectural principals. The prototype application will also use an existing vehicle communication interface (iVIC dlls) and the main VADIS2004 framework. Another objective is to test rich application functionalities (compared with current VADIS) by using modern web technology (FLASH).

4.2 Problem and Solution

Focusing exclusively on the PowerTools system certain problem definitions can be made. We need to perform advanced complex communication to Volvo vehicle, but we do not desire to download and install any applications on the client. Furthermore we would like to use minimum amount of bandwidth.

The solution to the distinct problems is to use autonomous FLASH applications. These applications are very small in byte-size therefore rapidly downloaded. (For arguments concerning the use of FLASH, see chapter 6.4.6). Each FLASH application is controlled by an XML based script (PowerScript) which is downloaded to the client. All the downloading process is automated, meaning that the user on client side only needs to “click on a link” on a website viewed in a web-browser. The implementation and use scenario of PowerTools is illustrated in Figure 9 below.

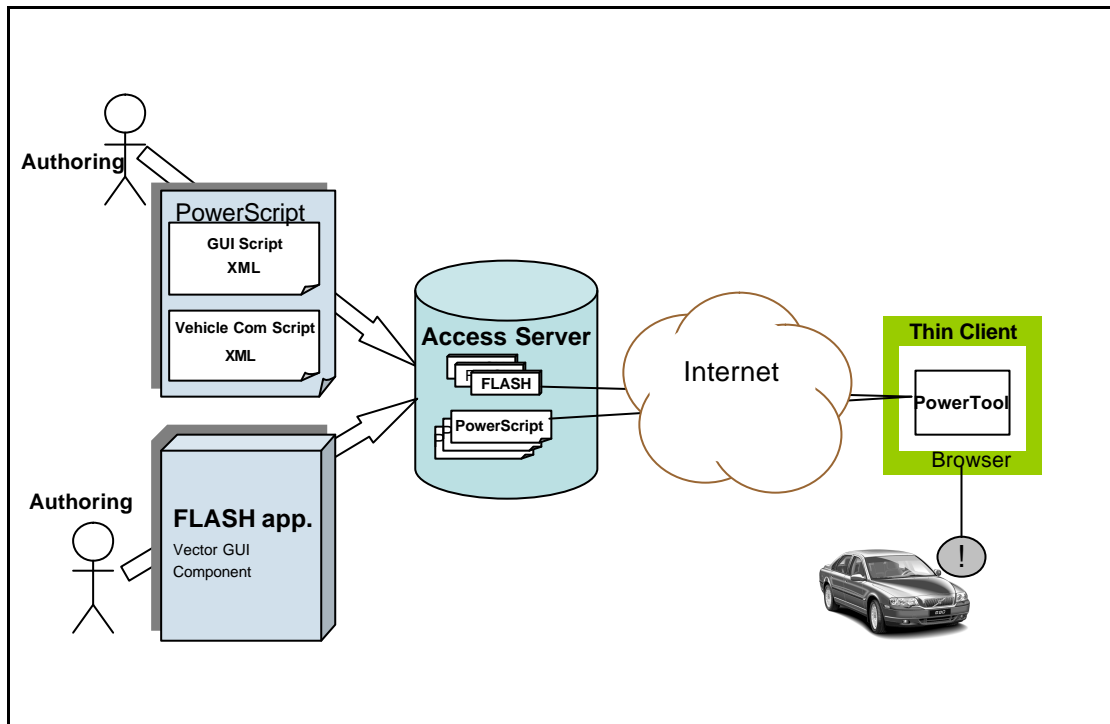


Figure 9 Deployment model of PowerTools components

The PowerTools author creates a FLASH application and place it on the Access Server. A PowerScript is the authored, by another or same author and then placed on the access server. Upon request from the user the FLASH application along with the controller script are downloaded to the client and can be used for vehicle communication. In many cases same FLASH application can be reused for different purposes by using different controller PowerScripts. For instance a meter FLASH application can show the running state of either the current value for inside temperature or the current value of the washer fluid level, depending on which PowerScript file is assigned to it. A PowerTool is therefore defined by the assigned PowerScript and not the actual FLASH file.

PowerTools and PowerScripts are described in detail in Appendix A.

4.3 PowerTools and mobility issues

The PowerTools system design implications are, among other factors also based on mobility issues concerning the user side of the system. Mainly two general cases have been taken in consideration. The cases are:

1. Application-user mobility
2. Vehicle mobility

The first case concerns mobility issues related to the user environment and the user her self. An example is the case where mechanics has to be able to be mobile in the workshop environment and around the vehicle for a diagnostic session. This case is made possible due to the fact that the client side of the PowerTools system is a so-called “thin client” and can therefore be deployed on small mobile PDA units such as iPAQ. Cases where the mobility is not local meaning that it is restricted to the workshop environment, The PowerTools system can be deployed with all parts (client and servers) into one powerful mobile unit like a TabletPC [19] or a laptop computer. This way the system allows for global mobility for the user independent of any infrastructure support.



Figure 10 – Mobility in the workshop environment

Case 2 deals with situations where the movement and mobility is based on the vehicle itself, for instance a diagnostic session on a vehicle while it is being driven or in case of remote diagnostics. This use case is made possible because of the fact that the PowerTools system is a Web-based solution. The PowerTools support for this type of mobility is highly dependent on an infrastructure supporting it.

How these mobility issues in different use scenarios where defined, and the development based on these issues, are further discussed in chapter 5.1.3.

5 Designing the system

In this chapter the theories for designing the system are presented. Then the main stages of the design process are further discussed.

5.1 Methodology

I have used mainly conventional techniques and methods typically used for these kind of projects. The project was divided in five parts:

- Field study and information gathering
- Analyzing and organizing of the gathered information/data
- Theoretical design implication of the system based on defined architectural principals.
- Development
- Evaluation / Testing

The process is illustrated in Figure 11. Below

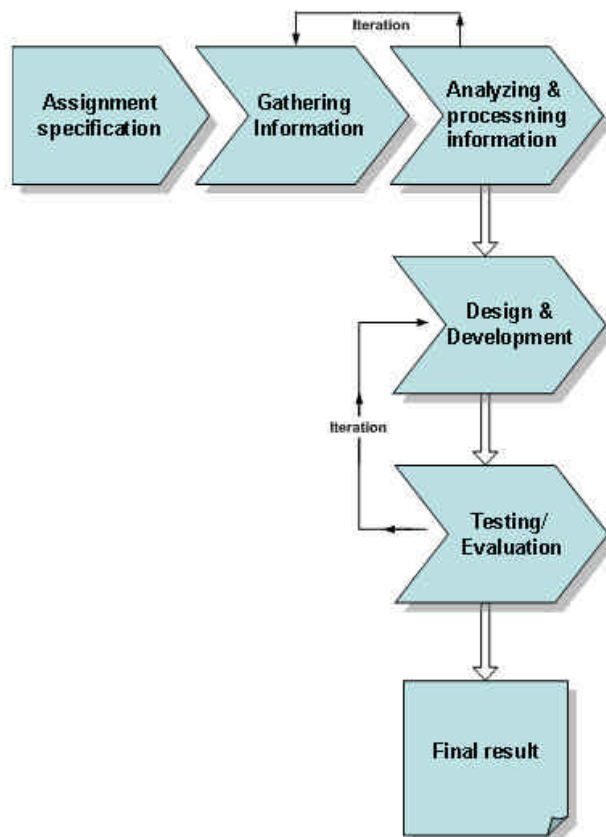


Figure 11 – Process model

5.1.1 Field study and information gathering

During the initial stages of the project (also before, as part of a student assignment at the IT-university in Gothenburg) I carried out field studies at Volvo Cars Workshops. The field studies were not carried out as conventional ethnographic studies, due to the fact those kinds of studies are very time consuming and deal with a bigger picture on a much deeper level. The field studies performed was more of the so called "Quick-and-Dirty Ethnography" studies type [10], meaning that the field studies were performed during a short limited time with focus on certain aspects and with a design goal in mind.

The plot of these studies was to gather information not based on any particular detail subject in the use of the diagnostic tools but to focus on documenting a full cycle of a diagnostic session for a car using VADIS. As part of the study I used a digital camera to photograph interesting subject used in the diagnostic sessions. I also performed interviews with some of the mechanics.

This type of information gathering is mainly used when you are looking for previously known problem issues, which in my case was about problems using today's Diagnostic tool at Volvo Cars; VADIS. By carrying out the field studies at Volvo Cars Workshop I could, during the coming design stage of my project, refer to the issues that I had discovered. However, the results of the studies were mainly used to give me better knowledge about the VADIS system and diagnostics at Volvo Cars in general.

Based on results from previously carried out field studies at the workshops by students at IT-university in Gothenburg, an extensive series of ethnographic fields studies at the workshops was carried out by a group within the VADIS NG project. Among results from these studies, were guidelines concerning GUI design.

In addition to the information gathered through the field studies, information was collected through VCCS documentation studies and consultation with the employees.

5.1.2 Analyzing and organizing the gathered information/data

When the information and data was gathered the next step was to organize them for future use in the development stage. The information was organized as follows:

- Use related information
- Information to build the necessary knowledge base for development
 - Existing hardware for Vehicle Communication.
 - Existing infrastructure (in development/test environment)
 - Setting up a connection to the Vehicle
 - Using today's VADIS
 - How to understand and interpret DII
 - How to use logs for tracing the DII communication
 - How to use other Vehicle Communication Apps such as: DHA
 - Using existing software components (iVIC)
- Data needed for the functionality of the prototype

- Chosen DII commands to be used
- Unique interpretation information for the DII

5.1.3 Design and Development

The Architectural design of the system was done based on factors concerning:

1. Development of the system
2. Implementation and deployment
3. The User perspectives

With these factors in consideration a set of case models was created. The models are presented bellow.

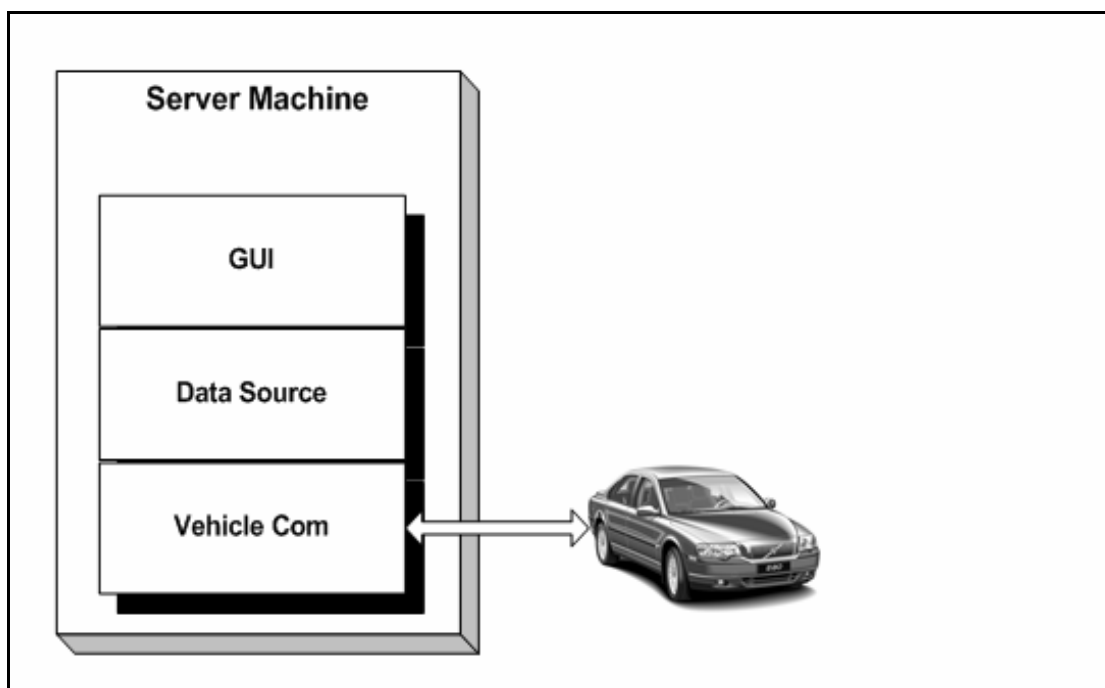


Figure 12 – Case 1

In case 1 illustrated above, the GUI, Data Source and the Vehicle Com are all in the same block, meaning that all the layers are implemented in one and same server machine. Development of such system presented in this case is done easily and the development process is thus short. The client in such system would be a so called “rich client”, because of the fact that the system is “all in one”. Seen from the user perspective this case locks the system to the same physical position as the car. The Vadis Cart that is used in Volvo Cars Repair Shops today is representative for case1.

From a mobile perspective this case allows high user mobility. Because of the fact that the system is so called “all in one” it can be deployed into powerful mobile units such as laptops or TabletPCs.

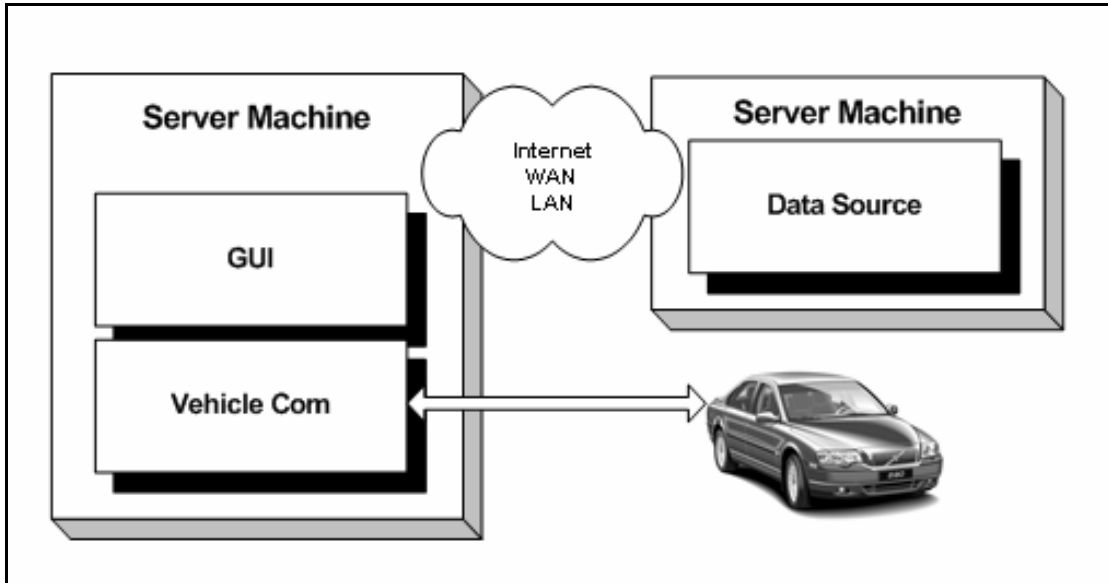


Figure 13 – Case 2

In case 2 the data source is placed out side on a separate block, meaning that the data is provided by an external data source positioned on a different data base server machine. In this case networking issues are to be considered because of the physical position of the data source. The user mobility is restricted due to the network infrastructure.

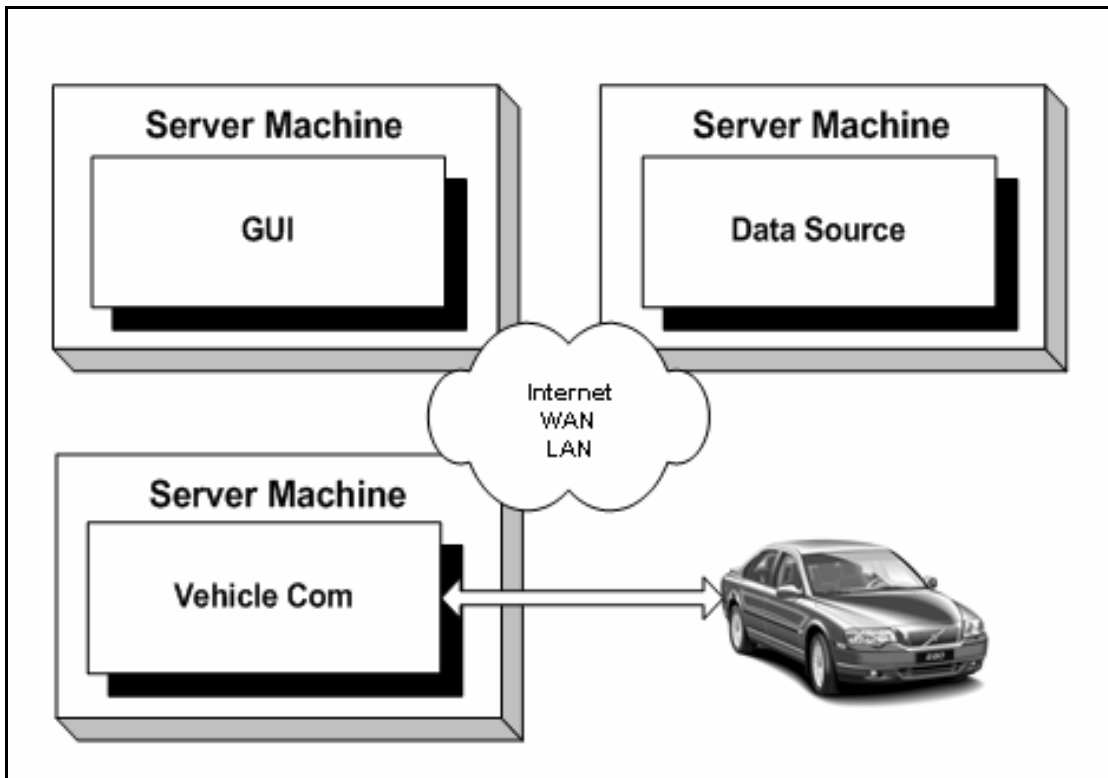


Figure 14 – Case 3

In case 3 all parts of the system are separated and a main server machine does not exist. The system presented in this case sets requirements on an infrastructure supporting it.

Because of the fact that the parts of the system are separated, including the GUI part, high user mobility as well as vehicle mobility is allowed in this case. However this case is, as mentioned before totally dependent on a supporting infrastructure.

For better understanding and visualization the cases are described through a users perspective as follows:

- Case1 – Mechanic X has the whole tool system in his Laptop PC. Upon a service call, he can travels to the location of the car and perform the needed diagnostics at place.
- Case2 – Mechanic X is currently performing diagnostics on a car. Certain data for a special ECU command has been updated and he can perform the diagnostics using that data without any further action from his side. If he was using his “All-in-one” machine (Case1) he would have needed an update kit CD/DVD.
- Case3 – Mechanic X is sitting in his “office” in Gothenburg (Sweden) and performing diagnostics on a car being driven or parked in Los Angeles (USA).
- Case3 – Mechanic X is lying under the car adjusting it while at the same time monitoring the related values in runtime, on his iPAQ PocketPC device.

Conclusion based on the presented cases

When designing the system architecture I took in consideration all the three cases as they all can accrue depending on existing infrastructure and hardware. Therefore the system had to be designed with a structure that would support all cases. A multilayered system architecture was the solution. Layered architecture is further discussed in chapter 6.2.

Furthermore, the System design is based on the theory of object oriented system design. One of the requirements for the prototype was that it had to be implemented in the main VADIS NG framework. Therefore the design of the power tool system had to follow the same design theories as the main system. The main VADIS2004 is being designed on a framework based on UML, which is developed by WM-Data and quality certified by Microsoft [18].

5.1.4 Evaluation

The evaluation method for the system was based on a number of tests and also this part was loosely based on the RUP (Rational Unified Processes) theories. So the D&D and the tests were done in an iteration process. More information about the evaluation process can be found in chapter 7.1.

5.2 Delimitations

The delimitations of this project were directly set by the definition of its goal. The goal of the project being to present a system architecture idea clearly sets the outline for what subjects that needs to be dealt with within the project.

An important aspect for demonstrating the functionality of a system is to present a red thread that can be traced through out the whole system. Hence there must exist a functional component within each part of the system. Although a system in use may have a large amount of data and components for its certain purpose, there is in fact no need for all that to be present to have a red thread.

So the limitations are mainly quantity delimitations. For instance for showing that a certain communication link within the system is functional it is enough to have a single feature/component that uses this connection.

Other delimitations consider the prototype application developed. The purpose of the application was as mentioned above to only demonstrate the system architecture idea; therefore it has not been of interest to emphasize the Graphical User Interface part of the application neither any specific diagnostics functions.

6 Development

Before initiating the development process it had to be decided what technologies were to be used. This decision is normally based on factors such as:

- Strength and power of the technology
- Modernity of the technology (being up-to-date)
- Adaptability and compatibility
- Development pace and financial costs

These are among some factors that usually are considered before deciding the technology to be used for the development. But in the case some of the technologies to be used for the development were already decided. The system developed within this project is a sub-system of the main VADIS2004 system that has been developed using mainly the Microsoft .NET technology. Therefore it has been important that also this sub-system was developed using the same technology for compatibility and adaptability reasons.

The next topics in this chapter describe the technologies and design theories used for development of the system.

6.1 Web Services

Web Services are platform neutral applications accessible using standard Internet protocols like SOAP, and metadata formats like XML. They are components, and represents black-box functionality [22]. The basic concept of Web Services is illustrated in Figure 15 below.

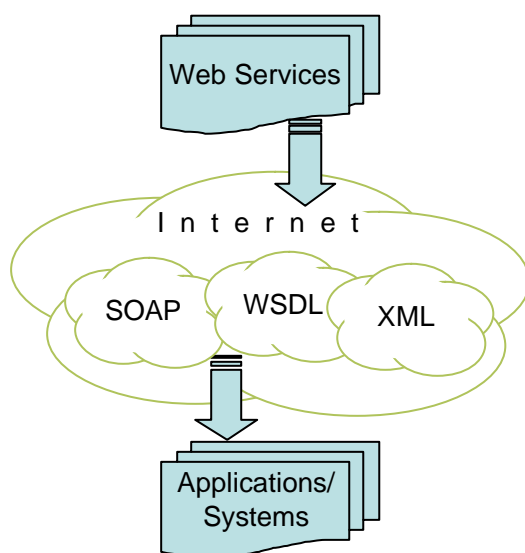


Figure 15 – Basic model of Web Services concept

XML Web services are the fundamental building block in the move to distributed computing on the Internet. Open standards and the focus on communication and collaboration among people and applications have created an environment where XML Web services are becoming

the platform for application integration. Applications are constructed using multiple XML Web services from various sources that work together regardless of where they reside or how they were implemented. XML Web services are successful for a variety of reasons, notably: They are based on open standards with multi-vendor support making them interoperable, and the technology used to implement them is ubiquitous.

XML Web services are built on XML, SOAP, WSDL, and UDDI specifications [22]. These constitute a set of baseline specifications that provide the foundation for application integration and aggregation.

When using Web Services for system integration design we make some additional definitions. We define a Web Service is as: “a computer supported task performed by one of the defined users of the system”. Web Services are described by Input Parameters, Action and Output Parameters. In some cases there are several methods that can be performed on one Service.

Defining Services can be somewhat disputable. What is a Service, a method or a process? However, functionality that can be easily understood and explained and possibly used by more than one actor should be defined as Services. Example: “Read Parts Information” means just that, and can be used by Volvo Car Company Dealers as well as by independent dealers. Further, it is possible to apply methods like “display”, “change” and “select” to “Read Parts Information” with parameters like “steering wheel”. “Read Parts Information can also fit into a repair process. Services should always be implementation-independent.

Researchers and professional dealing with redesign issues seem to be have common understanding that system design, on several levels, are moving towards a fractioned design. Fischer et al [9], states that “a construction kit with a large number of generally useful building blocks provides a good basis for redesign”, when discussing the redesign of monolithic system into object oriented system. Hence the use of web services in system design is appropriate when considering systems with the need of a layered architecture.

6.2 Layered Architecture

Generally it is desirable to use layered system architecture for complex systems. This is a good rule to follow when designing and developing web based systems.

By splitting the system into layers we achieve the following:

- The ability to implement the layers on different physical places. Meaning that for instance the data source layer can be placed on a different server than the application server.
- The possibility to replace, modify or update a certain layer without any major modifications to the rest of the system, unlike a system with a non layered structure that needs to be completely remade.
- Scalability possibilities. A layered structure is fundamental if the system requires being scalable.
- Possibilities for Interaction with other systems and also implementation of other sub-systems. In many cases a certain layer of the system could have an interaction with an external system. Implementing such an interaction is easily done, without any modification to rest of the system. For instance if the data-source layer needs to be provided data from another external system it can be implemented without the involvement of the presentation layer.

For this system I have presented the layered architecture design as shown in Figure 16.

The top layer which is the presentation layer consists of a Macromedia FLASH Application. The application itself can be divided into the two separate layers of GUI and application logic (Action Script code).

Beneath the Presentation layer I have placed the logical layers for communication. The left box represents the layer that holds the logic for interaction with a Volvo Vehicle. This layer contains the iVIC system which in itself is based on a layered structure. The iVIC system is Figure 16 further described in Appendix A.

For communication with the database providing the data for processing in the presentation layer, there is a Data Base Web Services layer. This layer contains the database and the database access object.

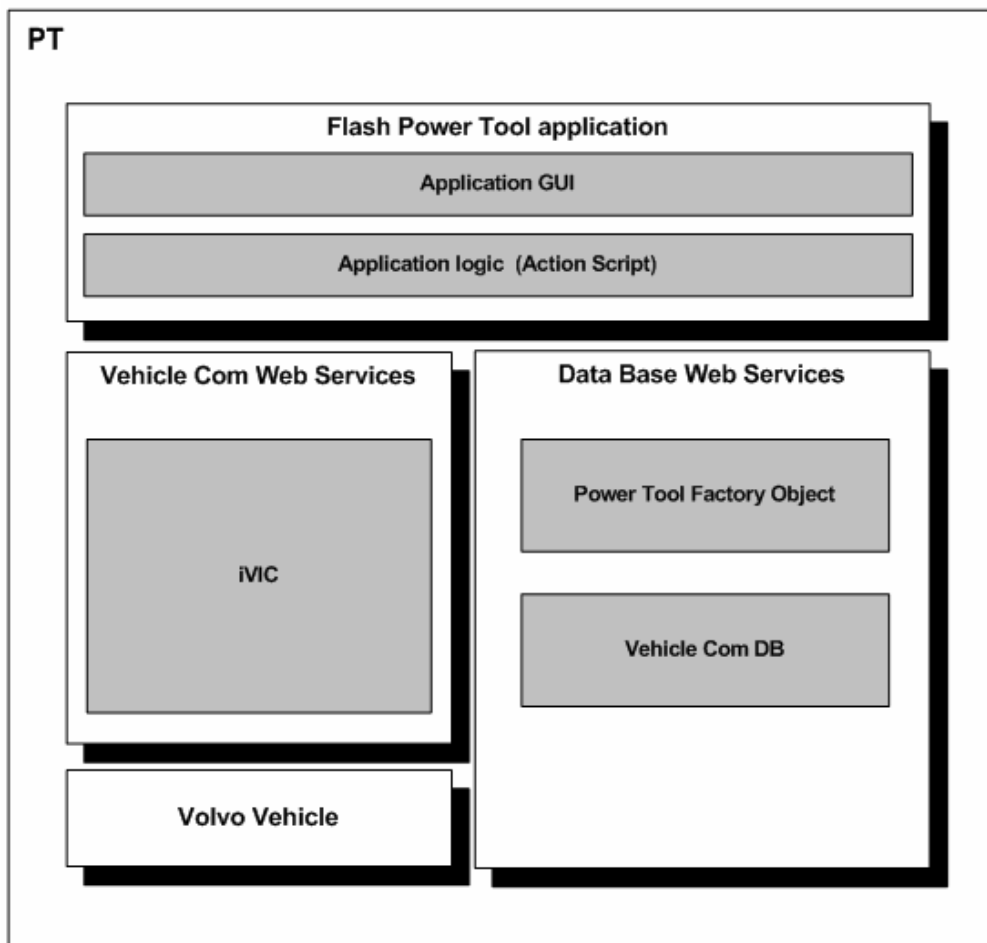


Figure 16 – Layered system structure of the PowerTool system

6.3 Programming in Microsoft .NET

6.3.1 .NET Language Model

.NET represents a whole new way of developing software. .NET is not tied to the Windows operating system. At the heart of the .NET platform is a common language runtime engine and a base framework. Most programmers are familiar with these concepts. Windows operating system itself can be thought of as a runtime engine and library. Runtime engines

and libraries offer services to applications, and programmers love them because they save time and facilitate code reuse.

The .NET base framework will allow developers to access the features of the common language runtime and also will offer many high-level services so that developers don't have to code the same services repeatedly. But more importantly, the .NET common language runtime engine sitting under this library will provide the technologies to support rapid software development.

Among features provided by the .NET common language runtime engine are:

- *Consistent programming model*

All application services are offered via a common object-oriented programming model, unlike today where some OS facilities are accessed via DLL functions and other facilities are accessed via COM objects.

- *Simplified programming model*

Developers no longer need to gain an understanding of the registry, GUIDs, IUnknown, AddRef, Release, HRESULTS, and so on. It is important to note that .NET doesn't just abstract these concepts away from the developer; in the .NET platform, these concepts simply do not exist at all.

- *Run once, run always*

Since installing components for a new application can overwrite components of an old application, the old app can exhibit strange behavior or stop functioning altogether. The .NET architecture separates application components so that an app always loads the components with which it was built and tested. If the application runs after installation, then the application should always run.

- *Execute on many platforms*

Today, there are many different flavors of Windows: Windows 95, Windows 98, Windows 98 SE, Windows Me, Windows NT® 4.0, Windows 2000 (with various service packs), Windows CE, and soon a 64-bit version of Windows 2000. Most of these systems run on x86 CPUs, but Windows CE and 64-bit Windows run on non-x86 CPUs. Once written and built, a managed .NET application (that consists entirely of managed code) can execute on any platform that supports the .NET common language runtime.

- *Language integration*

COM allows different programming languages to interoperate with one another. .NET allows languages to be integrated with one another. For example, it is possible to create a class in C++ that derives from a class implemented in Visual Basic. .NET can enable this because it defines and provides a type system common to all .NET languages.

- *Code reuse*

Possible because of language integration possibilities

- *Automatic resource management*

The .NET common language runtime automatically tracks resource usage. In fact, there is no way to explicitly free a resource

- *Rich debugging support*
- *Consistent error-handling*

In .NET, all errors are reported via exceptions—period. This greatly simplifies writing, reading, and maintaining code. In addition, exceptions work across module and language boundaries as well.

- *Deployment*

.NET components are not referenced in the registry. Hence, installing most .NET-based applications will require no more than copying the files to a directory, and uninstalling an application is done by deleting those files.

6.4 Macromedia FLASH

6.4.1 FLASH / FLASH Remoting version MX

A bandwidth friendly and browser independent vector-graphic object based technology is a short and simple description of FLASH [16].

With Macromedia FLASH, designers and developers can build portable, screen-agnostic, custom user interfaces for the device as well as for the desktop. Compared to device-specific programming languages, Macromedia FLASH is approachable and accessible, with a visual development environment. Developers can easily deploy existing content to Macs, PCs, and Linux machines in Internet Explorer and Netscape without recreating the content.

Macromedia FLASH Remoting MX provides a communications channel between Macromedia FLASH applications and a wide range of business logic and data from ColdFusion, Microsoft .NET, Java, and Simple Object Access Protocol (SOAP)-based web services. In this project the main frame work is built in .NET.

6.4.2 Using FLASH Remoting MX

Macromedia FLASH Remoting MX is an application server gateway that provides a network communications channel between FLASH applications and remote services.

6.4.3 Development with FLASH/FLASH Remoting MX

Because FLASH Remoting MX connects two distinct and separate runtime environments, FLASH applications with FLASH Remoting MX are built in two programming languages, ActionScript and the programming language of the application server which in this case is C#.

Because of the separation between the client and server environments, the development of FLASH applications using FLASH Remoting MX should be separated in different parts. In traditional HTML-based web applications, responsibilities usually fall into two general roles, designer and developer. The designer creates the HTML user interface, and the developer creates the application server logic.

In FLASH development using FLASH Remoting MX, it's useful to organize development parts as server-side development, client-side development, and client-side design. The client-side design creates the FLASH user interface, including layout, animation, and effects. The client-side development results in the ActionScript to connect to the remote service and handle the results. Finally, the server-side development is the business logic on the application server to serve as the remote service.

In this project the development can be divided in the following stages/parts:

- Design of FLASH Smart Tools GUI Components. Using Action Script for GUI functionalities.
- Programming ActionScript for business logic and Web Service interaction
- Development of .NET based Web Services for vehicle communication and data base access.

The following figure illustrates a simplified representation of the FLASH Remoting architecture:

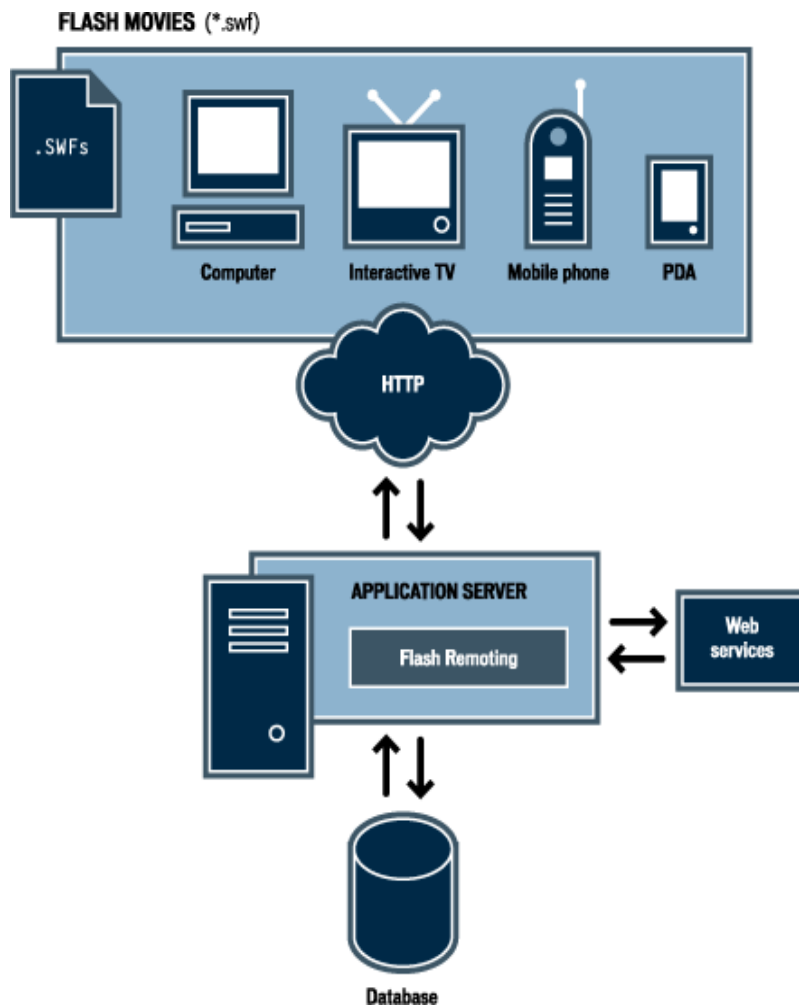


Figure 17 – FLASH Remoting architecture

6.4.4 FLASH Remoting MX and .NET

Macromedia FLASH Remoting MX exposes ASP.NET technologies as remote services to FLASH applications, which are accessible as ActionScript functions.

A variety of Microsoft .NET technologies can serve as remote services, including ASP.NET pages, web services, and assembly methods.

A FLASH developer writes ActionScript that uses a library of functions called Net Services to connect to a remote .NET server, get a reference to the remote service, and invoke the remote service's functions.

To transport messages, FLASH Remoting MX uses a binary message format called Action Message Format (AMF) delivered over HTTP and modeled on the Simple Object Access Protocol (SOAP) used in web services implementations. AMF is smaller and faster than standard SOAP, and is purely asynchronous and event-driven. It can send a variety of data types, including Record Sets, primitives like integers, strings, XML documents, and dates across the Internet using HTTP.

The FLASH Remoting gateway acts as a front controller on the ASP.NET runtime that handles the conversion of data types from ActionScript to the .NET Common Language Runtime (CLR) and so on. When the gateway receives a service request, the request passes through a set of filters that handle serialization, logging, and security before arriving at a service adapter designed to handle the service and invocation type.

FLASH Remoting MX contains four service adapters for .NET technologies:

- ASP.NET adapter
- ADO.NET data binding adapter
- Web services adapter
- Assembly (*.dll)

When embedded in ASPX pages with other server controls that render HTML, a FLASH application that uses FLASH Remoting MX becomes part of the client layer of a .NET application.

FLASH Remoting MX can be used as a custom server control in ASP.NET pages in .NET Web Form applications, or as a namespace in .NET DLL files, code-behind class files, and web services. A .NET assembly (FLASHgateway.dll), located within the local assembly cache of your ASP.NET application, provides the FLASH Remoting functionality.

6.4.5 Calling web services from FLASH

Using the FLASH Remoting web service adapter, we can call web services from FLASH that is described by the Web Services Description Language (WSDL). We must first generate a local web service proxy to interact with web services. After we create the proxy, the ActionScript in the FLASH application can then invoke web service methods through the proxy, which handles sending and receiving Simple Object Access Protocol (SOAP) messages with the remote web service.

In .NET, we can generate proxy assemblies with the WSDL Tool (wsdl.exe). FLASH Remoting MX for .NET also uses the WSDL Tool to generate SOAP proxies for web services automatically from valid WSDL, either local or remote. FLASH Remoting MX is not restricted to .NET-based web services meaning, any WSDL-described web service is available to FLASH Remoting MX. And can be accessed.

This may be of great importance if the application designed is depended on external systems.

6.4.6 Why FLASH?

FLASH technology has till now (because of limited features) been used mainly for used for rich dynamic graphical presentation purposes. Therefore use of actual web applications developed in FLASH has been rather unconventional. But today the case is different, with the introduction of Macromedia FLASH MX and related Macromedia internet technologies such as FLASH Remoting MX. The technology is more powerful today and allows integration and use of other powerful sources written in for example C# in ASP .NET and remote Web

Services. Among FLASH features that are desirable is its dynamic features when network connection interruptions occur.

Because of the fact that a FLASH application is automatically downloaded to the client upon access to the holder web site, there is no installation process when using FLASH applications. There are other ways to go around the installation process, using windows programmed ActiveX components is a way but development process in FLASH is much faster and all of issues concerning security barriers related to the download and use of ActiveX controls are bypassed when using FLASH.

The deciding factors for use of FLASH technology for this specific task (PowerTools applications) can be summarized as follows:

- Fast development process
- Rich dynamic graphics features
- Low bandwidth use
- Easy to deploy
- No installation on client side

6.5 Problems and Issues

There were a number of technical problems and issues during the development process using the previously mentioned technologies. But the issues were mainly related to the fact that both of the technologies used for the development are relatively new technologies.

Macromedia FLASH Remoting MX was in fact introduced to the market previously this year and I did encounter a major problem with this engine. The problem can be described as follows:

FLASH Remoting could not access the chosen web service proxy and set up the connection objects. Any effort for doing so did result in the following error message: *“COULD NOT CONNECT TO THE PROXY, CHECK IF THE MICROSOFT .NET FRAMEWORK SDK IS INSTALLED”*. This was in fact a frustrating problem due to the fact the .NET SDK was installed on the development computer.

The problem was however solved by uninstalling the existing .NET SDK and installing the latest version of the SDK.

Another problem similar to the one above accrued when a change was made in the .NET web service that FLASH Remoting connected to. Macromedia FLASH Remoting creates a set of .dll files upon connection to the web server proxy. There is a certain dll file for each web service that the application uses. The problem in this case was that these .dll files didn't update upon change in the actual web service. This issue was not fully cleared but according to the information provided by other FLASH Remoting developers the reason for the problem can be related to the caching properties of FLASH Remoting. A difficult way to clear this problem was to reinstall FLASH Remoting.

7 Evaluation / Validation

This chapter will discuss the evaluation and validation process of the PowerTools system. Although validation on some level has been done, the process is however not over. The system will be further evaluated and validated on other occasions in the future when VADIS2004 has been released.

7.1 Evaluation

The evaluation process of the prototype and the system architecture developed in this project has been part of an iteration process.

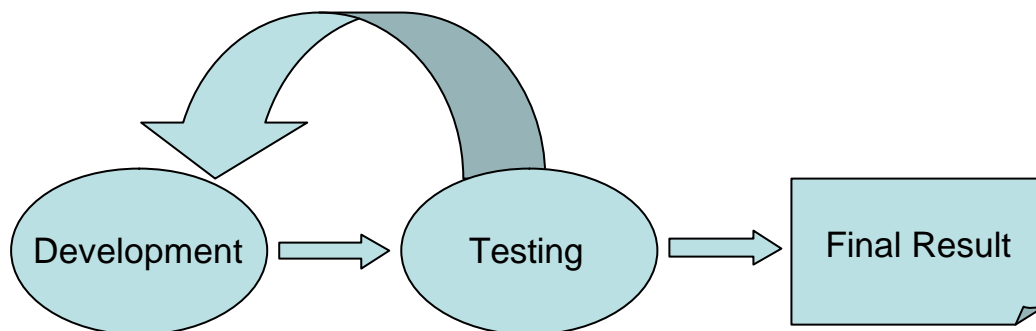


Figure 18 – Evaluation process

7.1.1 Testing

The testing is an essential part of the evaluation process. To be able to fully evaluate the system, I specified a set of tests based on the objectives of the project.

The results from the tests were to deliver an answer to the following questions:

1. Is the prototype FLASH application functional by itself (GUI functionality without connection to the rest of the system)?
2. Does the .NET Web Services work?
3. Does the FLASH application work together with the rest of the system?
4. Can a red thread be traced through out the whole system by performing a likely diagnostic scenario?

A positive answer to all of these questions after the tests has been carried out for all of the possible infrastructural scenarios (as discussed in chapter 5.1.3) means that the result of the project can be considered as satisfying from a technical perspective.

For this purpose I did deploy the system according to the specified infrastructural scenarios and carried out the test. Obviously there was problems and issues encountered when switching to more physically fractured infrastructural scenarios. But by localizing the problems through traditional fault tracing techniques in the iteration process, the bugs could be eliminated.

The system was tested in the infrastructure scenarios presented in Figure 19.

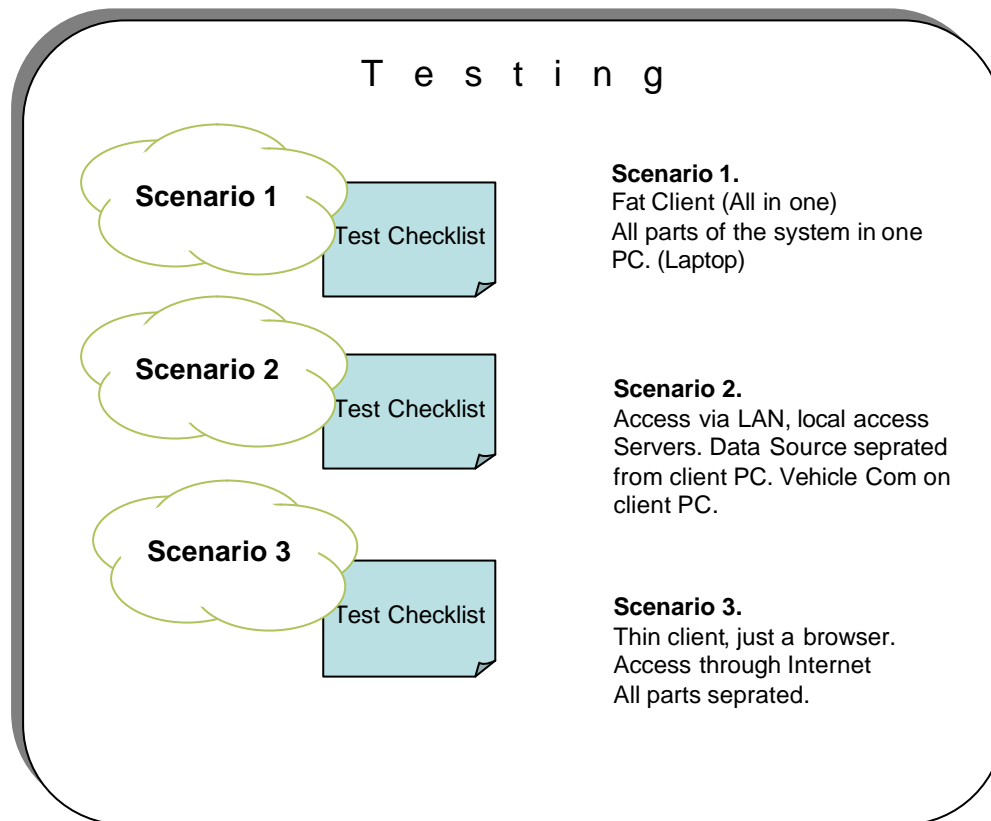


Figure 19 – Tests based on different scenarios

7.1.2 Conclusion


The outcome from the tests indicated that the layered structure of the system in fact did show proof of flexibility, adaptability, scalability and mobility properties of such system architecture.

Depending on the case of use, the system can be deployed in different ways. For instance where infrastructure supporting reliable and fast Internet exists the installation process of the client application is eliminated hence the client side only exists of a PC running a Web Browser (Internet Explorer with Macromedia FLASH Player plug-in).

7.2 Validation

Because of the fact that the system and the application developed are prototypes, the traditional validation process, which would be to go back to initial starting point where the studies on the users took place, is not possible. When designing a prototype the finished “product” cannot be sent in to the “market” for validation.

However, parts of the system such as the GUI can be considered as validated. The GUI of the prototype was based on GUI suggestions and guidelines that have been validated by chosen user groups. These GUI guidelines were part of the results from extensive ethnographic studies carried out at Volvo Workshops by a group from VCCS.



The developed system in this project is a part of the 0.1 release of VADIS2004. At Volvo Cars Customer Service there has been several so-called “Reference-group-meetings” scheduled. On the latest Reference-Group-Meeting occasion the progress of the VADIS NG project was discussed and informed. At this meeting the PowerTools system was demonstrated for the Reference-Group that consisted of 28 people from different Volvo Cars units around the world with expertise in the Vehicle Service area. For demonstration, a Volvo vehicle positioned in the Workshop area was remotely controlled and monitored. The response from the reference-group was unanimously positive.

8 Summary

8.1 Objectives and achieved results

The main objectives of this master thesis project was to present a technical reference base for future development and implementation. This was done by developing a functional prototype system illustrating the presented ideas. This project was carried out as part of the ongoing VADIS NG project.

8.1.1 The System architecture

It was required that the system should be based on such architectural principals that it could be deployed in various infrastructure scenarios predefined for the new version of Volvo Cars diagnostics system, VADIS NG (Volvo Aftersales Diagnostics System Next Generation).

A layered system architecture was the solution to meet the requirements defined. The system introduced the use of .NET based web services and rich applications, “PowerTools”, developed in Macromedia FLASH and new XML-based Scripts “PowerScript” for definition and control of the “PowerTools” applications. This layered architecture allowed the system to be configured differently based on user mobility, user preferences and user task.

8.1.2 The Prototype

A functional prototype application was developed for demonstration of the system. The application developed in Macromedia FLASH using .NET web services developed in C# demonstrated simple diagnostics sessions using PowerTools monitoring and controlling different features in the vehicle. The prototype proved that the technologies used for the development of the applications were suitable.

An interesting fact concerning the prototype was that, when the main frame of the application was in place and functioned, development and implementation of new PowerTools applications was a relatively easy task and could be done rather fast.

Acronyms

ACM	Association for Computing Machinery
ADO	ActiveX Data Object (Microsoft)
ASP	Active Server Page(s) (Microsoft)
CAN	Car Area Network
CARB	California Air Resources Board
CEM	Central Electronic Module
CLR	Common Language Runtime (Microsoft .NET)
COM	Component Object Model (Microsoft)
CPU	Central Processing Unit (computer)
CSCW	Computer Supported Collaborative Work
D&D	Design and Development
DB	Database
DHA	Diagnostic Host Application.
DII	Diagnostics II
ECU	Electronic Control Unit
EPA	Environmental Protection Agency (US government)
GUI	Graphical User Interface
HCI	Human Computer Interaction
HTML	Hyper Text Markup Language
ISO	International Organization for Standardization
LAN	Local Area Network
MOST	Media Oriented Systems Transport
NEVIS	New Editor and Vehicle Information System
OBD	On-Board Diagnostics
OS	Operating System
PDA	Personal Digital Assistant (electronic handheld information device)
R&D	Research and Development
RUP	Rational Unified Process (software engineering process)
SAE	Society of Automotive Engineers
SDK	Software Development Kit
SOAP	Simple Object Access Protocol (XML protocol)
TCU	Telematics/Telecommunications Control Unit
UDDI	Universal Description, Discovery and Integration
UML	Unified Language Model
URL	Uniform Resource Locator (world wide web address)
VADIS	After sales and Diagnostics Information System
VCCS	Volvo Cars Customer Service
VCT	Volvo (Vehicle) Communication Tool
WAN	Wide Area Network
WS	Web Service
WSDL	Web Services Description Language

References

1. Association for Computing Machinery (ACM) Website 2002-12-15
<http://www.acm.org>
2. B&B Electronics 2002 OBDII Home Page <http://www.obdii.com>
3. California Air Resources Board , Website 2002-12-15
<http://www.arb.ca.gov>
4. Dahlbom, B. (1996) The New Informatics. Scandinavian Journal of Information Systems, Vol. 8, No. 2, pp. 29-48, 1996.
5. Dahlbom, B. and Ljungberg, F. (1998). Mobile Informatics. Scandinavian Journal of Information Systems, Vol. 10, No. 1-2, pp. 227-233, 1998.
6. Dix, A & R. Beale, (1996). Introduction, In Remote cooperation. CSCW issues for mobile and teleworkers, Springer-Verlag, London.
7. Environmental Protection Agency, Website <http://www.epa.gov>
8. Fagrell, H (2000) Mobile knowledge (PHD thesis, Gothenburg University, Informatics Inst.)
9. Fischer, G., Lemke, A. C., Rathke, C., From design to redesign, Proceedings of the 9th International Conference on Software Engineering March 1987
10. Hughes, J., King, V., Rodden, T. and Andersen, H. (1994). Moving Out from the Control Room: Ethnography in Systems Design". In Proceedings ACM Conference on Computer Supported Collaborative Work, CSCW'94, pp 429-439, 1994, Chapel Hill, NC, USA.
11. Hughes, J., Randall, D. and Shapiro, D. (1992). Faltering from Ethnography to Design. In Proceedings of the conference on Computer-supported cooperative work 1992, Toronto, Ontario, Canada, pp. 115-122., 1992, ACM Press, New York, NY, USA. ***
12. IEEE Website 2002 <http://www.ieee.org>
13. Innovation of IT Use in Mobile Settings: IRIS'21 Workshop Report Steinar Kristoffersen, Fredrik Ljungberg
14. International Organization of Standardization, Website 2002.12.15
<http://www.iso.org>
15. Kristoffersen, S. and F. Ljungberg (1998) Representing modalities in mobile computing. In B. Urban, T. Kirste and R. Ide (eds) Proceedings of Interactive applications of mobile computing, Fraunhofer Institute for Computer Graphics, Germany.
16. Macromedia, Inc 2002 Macromedia Flash MX and Flash Remoting MX Support Website, 2002-12-15
<http://www.macromedia.com/support/flash>
http://www.macromedia.com/support/flash_remoting/

17. Microsoft Co 2002 .NET Website <http://www.microsoft.com/net/>
18. Microsoft Co. Press release 2002-06-10
http://www.microsoft.com/sverige/pr/articles/Pressmeddelande_542.asp
19. Microsoft Co. Windows XP Tablet PC, Website 2002-12-15
<http://www.microsoft.com/windowsxp/tabletpc/default.asp>
20. Society of Automotive Engineers, Website 2002.12.15 <http://www.sae.org>
21. Telematics Research Group at Viktoria Research Institute in Gothenburg, Sweden 2002-12-15 (Consultation with researchers)
22. Web Services references on the Internet, e.g. <http://www.uddi.org>,
<http://www.w3.org>
23. World Wide Web Consortium (W3C). eXtensible Mark-up Language (Technical Reference). 2002-12-15 <http://www.w3.org/TR/REC-xml>



Appendix A

- Technical Specifications -

Technical Specifications

System Architecture

The main concept of the system is illustrated in Figure 20. Each block in the system can be positioned on a separate physical position depending on the infrastructure. This means that the connection links between the blocks can in the cases needed go through the Internet and in other cases through LAN networks. The different parts of the system are described separately.

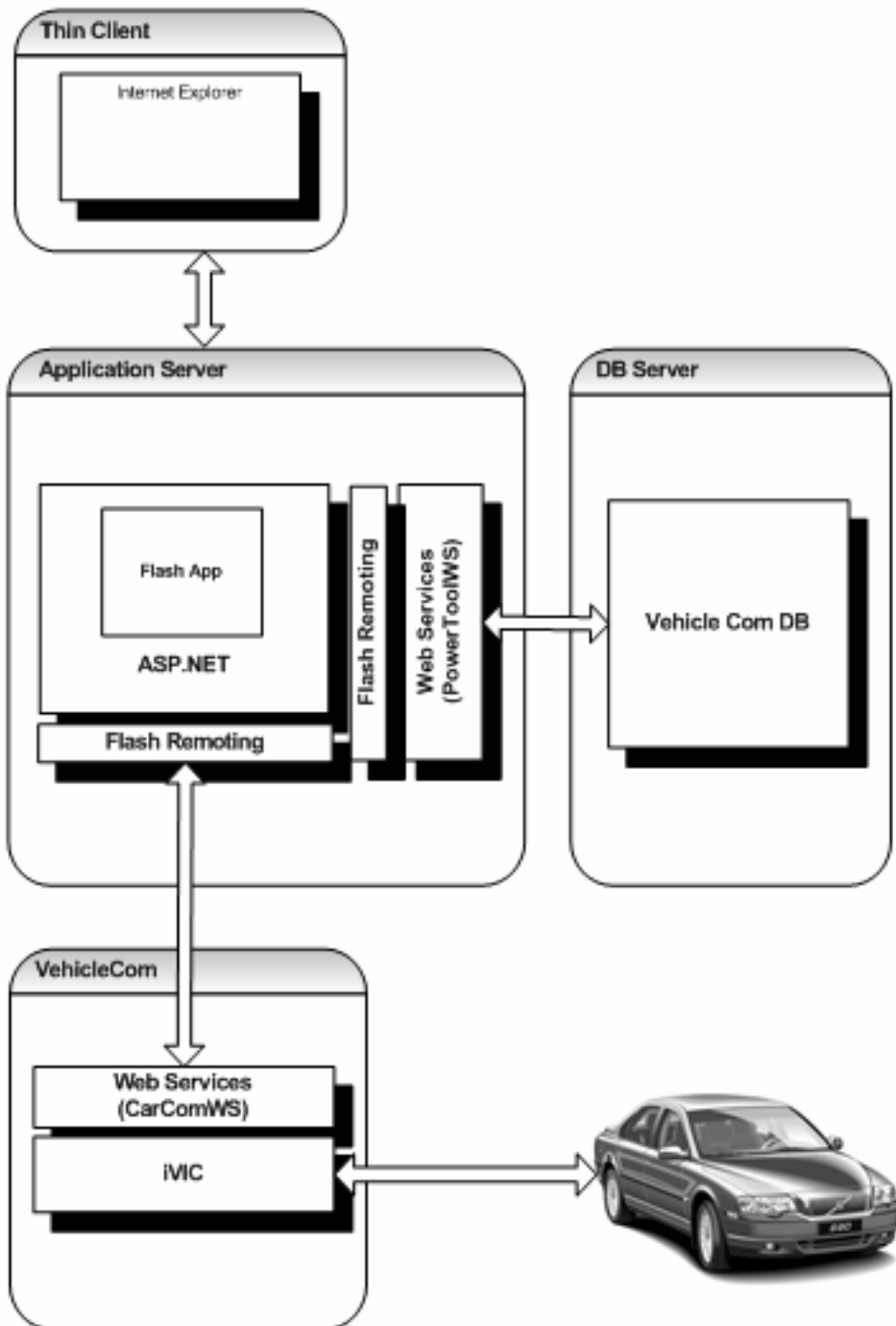


Figure 20 – PowerTools system design model

Vehicle Com Server

The vehicle communication is today limited to use of certain hardware (VCT2000, OBDII-LINK). The iVIC communication interface has been used for communication to the vehicle through a VCT2000 box. The VCT2000 box is connected to the COM-port of a PC. The connection model is illustrated in Figure 21 below.

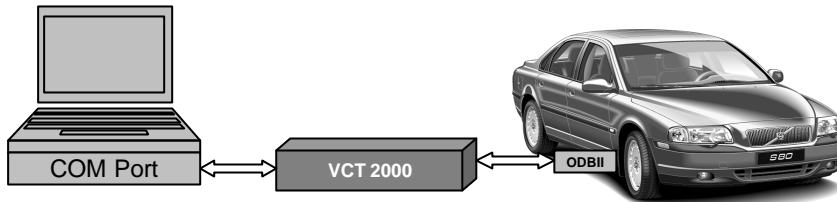


Figure 21

The iVIC vehicle communication interface is implemented as .NET Web Services making it possible to access the vehicle through Internet. Of course the host computer for the iVIC Web Services must have an IIS. This computer will then function as a vehicle-connected-server. The idea is that in the future when the hardware support exists all the parts in the Vehicle Com block in the main system model will be merged into one basic hardware component. This hardware component would then be integrated within the Vehicle ECU network.

Access Server and DB Server

The Access Server and DB Server can and will most probably be on the same machine in a workshop LAN.

The Access Server machine hosts the ASP.NET web site holding the FLASH application. The FLASH PowerTool files however can be placed on other server machines than the access server. Information about the physical location on the FLASH files along with the assigned PowerScript files is stored as a PATH in the PowerTool object.

The PowerTool object is stored in a Microsoft SQL Server on a remote DB server machine. The FLASH application on the Access Server communicates with the DB Server machine and the Vehicle Communication Server machine through FLASH Remoting and a set of Web Services. The web services are described in “Classes and Methods” chapter.

Client

The system is designed in such way that there is no need for installation of any application on the client side. The client side consists only of a machine running an Internet Browser with a FLASH Player plug-in. To run the application the user on the client side only requests access to an URL, in this case the address for the application web site on the Application Server.

Because of the fact that the client is a so called “thin client” the client machine can vary depending on different use scenarios. The client machine can for instance be an IPAQ PDA running Pocket PC and Internet Explorer.

Classes and Methods

.NET Vehicle Communication Web Service

For communication to the vehicle I have developed a .NET Web Service. The Web Service is programmed in C# and uses the iVIC for communication to the Volvo vehicle.

The iVIC system dll files were initially converted into .NET Framework compatible dll files then implemented into Web Service Class. The Web Service Class named **CarComWS:iVICservices** contains the following methods:

- *iVICservices()*

In: None

Out: None

Description:

This is the Constructor method of the class. When an instance of the class is made the method will be called. Within this method all the needed objects for communicating with the vehicle are initiated. One of objects that need to be initiated is the Vehicle Com Object. The settings property of this object must be set so that a communication link to the vehicle can be set up. But the settings can be changed through the SetMySettings() method.

- *SetMySettings(string strXMLsetting)*

In:

AN XML string with the settings Below is an example of the setting XML.

```
<VehicleComm>
  <VCT2000>
    <CommSettings CommPort='1' BaudRateVCT='115200' />
  </VCT2000>
</VehicleComm>
```

Out: None

Description:

This method is used to set the setting property of the Vehicle Com Object.

- *SendCommand(string strXMLcommand, string keepOpen)*

In:

strXMLcommand is a XML string that holds the instructions that will be sent to the vehicle.

keepOpen is the flag that controls if the connection port should stay open or not. The flag is set to "true" when the command is to be directly followed by other commands.

Out: String responseXML

An identical XML string to the in-parameter XML string, the difference is that the "ReadOutValue" attribute under the "Response" node does have a value assigned to it.

Bellow is an example of such a response.

```

<VehicleCommunication>
  <Collection Name="Instruction">
    <Instruction Method="DiagService"
      Protocol="D2" OrderPosition="1">
      <MethodData ECUAddress="01" BusType="1">
        <Collection Name="Services">
          <Service Name="BCM: Read CAN config part"
            Value="B9F5" IgnoreAckNackErr="1">
            <Response Id="" ReadOutValue="F9F50009456083202045FFFFFFFF
              0009459421202045FFFFFFFF"/>
          </Service>
        </Collection>
      </MethodData>
    </Instruction>
  </Collection>
</VehicleCommunication>

```

Description:

This is the Constructor method of the class. When an instance of the class is made the method will be called. Within this method all the needed objects for communicating with the vehicle are initiated. One of the objects

- **GetNodes()**

In: None

Out: String responseXML.

AN XML string of same kind as a regular response XML string from the SendCommand method.

Description:

This method returns an XML string that contains answer to the D2 question "B9F0" to all possible ECU nodes in the vehicle. The ECU nodes response to the question is the Hardware version number followed by the Software version number. This response is sent back as an attribute to the response node in the XML.

Obviously only the nodes that exist and are functional will respond to this method.

Consequently the GetNodes method is used to retrieve all available nodes the identification information about them.

- **GetVehicleProfile()**

In: None

Out: XML String

Description:

The XML string returned by this method contains a set of nodes, where each node holds information on the different vehicle parameters defining the vehicle profile. For Example Model Year etc.

- **Stop()**

In: None

Out: String noError

Description:

This method is the general stop method for all ongoing diagnostic services. If total stop is

accrued upon method call a TRUE response is returned, otherwise FALSE, meaning that there has been an error.

iVIC iVehicleCommunicator

iVIC is a common interface to execute XML based instructions towards Volvo vehicles. It supports SWDL and D2 protocols. The structure is layered in such way that it enables a high flexibility when it comes to managing different types of communication hardware.

The calling application needs to decide what protocol to use and what to send to a given hardware.

In this project the client application which is a web application developed in Macromedia FLASH will use the iVIC for communication with the vehicle. The iVIC Components will be accessed through a collection of Web Services on a .NET framework. The iVIC has been developed by Volvo IT AB exclusively for Volvo Cars Co.

iVIC consists of the following component objects:

- ***iVehicleCommunicator***

The IVehicleCommunicator interface provides methods to setup, initiate and communicate towards the Volvo Vehicle.

The message sent to the vehicle through the IVehicleCommunicator object is an XML script. An example of such XML script message for vehicle instructions with DII could look like this;

```
<VehicleCommunication>
  <Collection Name="Instruction">
    <Instruction Method="DiagService" Protocol="D2" OrderPosition="1">
      <MethodData ECUAddress="01" BusType="1">
        <Collection Name="Services">
          <Service Name="BCM: Read CAN config part"
            Value="B9F5" IgnoreAckNackErr="1">
            <Response Id="" ReadOutValue=""/>
          </Service>
        </Collection>
      </MethodData>
    </Instruction>
  </Collection>
</VehicleCommunication>
```

The D2 response from the vehicle is placed in the ReadOutValue attribute. To execute multiple D2 services, the services are added within the collection node in the script.

- ***IUtilityHandlerFactory***

This interface provides access to notifications and logger utility objects. This object enables several objects in several layers to send notifications of application events and to send trace statements for logging to the same receiver.

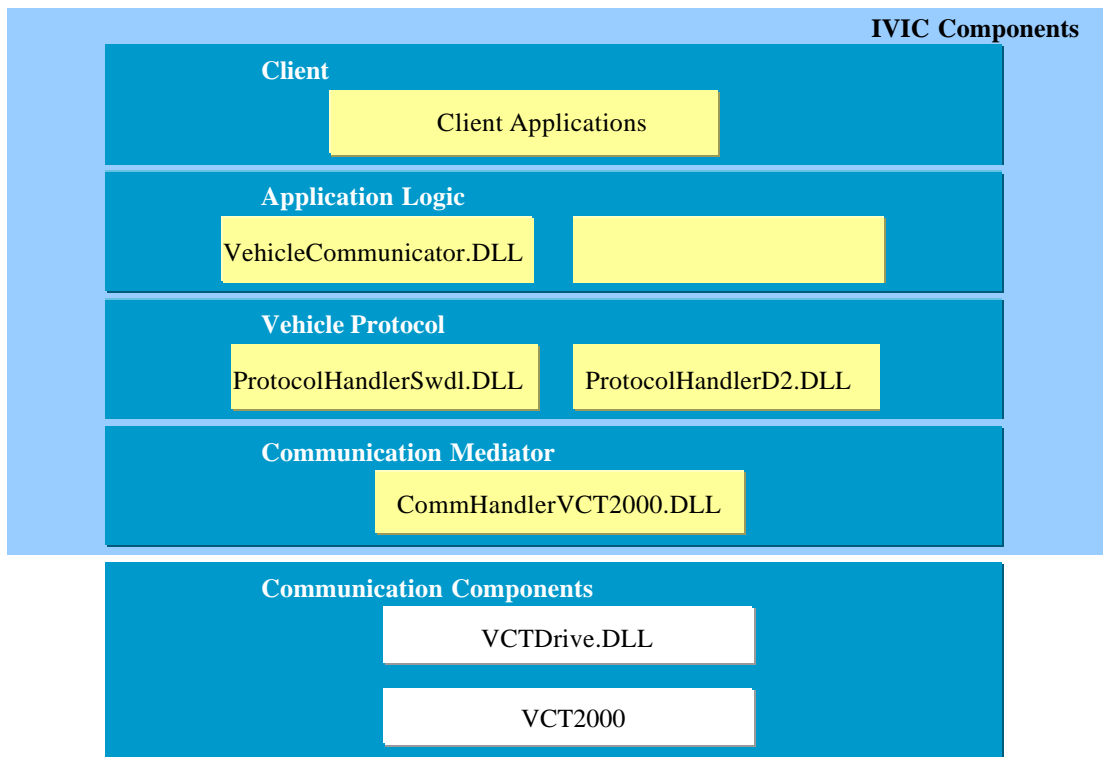
- ***ILogger***

The ILogger interface provides access to logger objects.

By using objects that has implemented this interface trace messages can be sent to different devices (i.e. file, screen, printer).

- ***ILogHandler***
Main handler of logging mechanism. All trace messages are routed through this object.
- ***IFileLogger***
The IFileLogger interface provides methods to put trace messages to files.
- ***IListener***
The IListener interface provides methods for receiving notifications events of the application progress.
- ***INotifier***
The INotifier interface provides methods for sending notifications of the application progress.
- ***IProtocolHandler***
Interface towards different vehicle specific protocols (SWDL, D2, SWDLNG).
- ***ICommHandler***
Interface towards different hardware layers (VCT2000, J2534, others).
- ***IcommHandlerFactory***
The ICommHandlerFactory interface provides access to commhandler objects. This object enables several objects to use the same commhandler object, previously created.

The implementation of the component objects in a layered structure is shown below.



.NET Database Web Service

The database connection in this system is managed through a Web Service class named: **PowerToolWS:DataBaseService**

The connection to the data base is managed by this class through the PowerToolCollection object which is provided by the persistent framework.

- **DataBaseService()**

In: None

Out: None

Description:

The constructor method of this class, initializing the database connection objects

- **GetPTByID(string ecu)**

In: A string containing the ECU id, which is based on the ECU hardware and software version number

Out: String PTcollectionXML

Description:

This Method is used to retrieve the PowerTool objects in the data base with the matching ECU-Id attribute. This means that for a certain ECU a collection of related PowerTools objects are found. After the search is done and the objects are retrieved, the objects are transformed into XML nodes containing node attributes holding the different PowerTool object attribute values. The method simply returns a string XML which is a collection of PowerTool nodes. Below is an example of such PowerTool collection XML.

```
<tools>
  <tool name="HeadLightTester" desc="Tests the Head Lights"
    FLASHFile=" HeadLightTester.swf" powerXML=" HeadLightTester.xml"
    fg="35">
  </tool>
  <tool name="EngineSpeedReader" desc="Monitor Engine Speed" FLASHFile="
    EngineSpeedReader.swf"
    powerXML=" EngineSpeedReader.xml" fg="38">
  </tool>
</tools>
```

PowerScript / PowerTools

A PowerTools is defined as a combination of a Macromedia FLASH file and a PowerScript XML file. The FLASH file is the actual application and the XML file is the code that controls the functionality of the application. Some FLASH application files can be reused for different PowerTools only by assigning them a different PowerScript file.

FLASH Application Files

An example of a FLASH application file developed in this project is ReadEngineSpeed.swf. The file is in swf format which is the FLASH movie file format. The FLASH application file needs to:

- Hold necessary GUI functionality
- Set up and use defined Web Services
- Hold PowerTool logic (The logic for interpretation and use of the controller PowerScript)

The programming language in FLASH is called Action Script and its syntax is very similar to Java/JavaScript syntax. Below is a sample Action Script code for setting up a connection to a Web Service and calls a certain method.

```

if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayUrl("http://localhost/FLASHremoting/gateway.aspx");
    gateway_conn = NetServices.createGatewayConnection();
    carComService = gateway_conn.getService("http://localhost/CarComWS/iVICservices.asmx?wsdl", this);
    dBaseService = gateway_conn.getService("http://localhost/PowerToolWS/DataBaseService.asmx?wsdl", this);
}
carComService.GetNodes();

```

Extract 1 – Code in Action Script

A simple PowerTool FLASH application file is very limited in byte size. The average size of the files created in this project was approximately 20 KB.

XML PowerScript Files

A PowerScript file is an XML file loaded by the FLASH application and then parsed. The PowerScript holds information that controls the functionality of the PowerTool and also GUI related information. Shortly the PowerTool is defined and controlled through the PowerScript.

Below is an example PowerScript file. This PowerScript (ReadEngineSpeed.xml) is the controller script file for a PowerTool used to monitor vehicle engine speed. The PowerTool is illustrated in Figure 23 in the next chapter.

```

<PowerScript version="1.0">
  <PowerTool Id="" Name="EngineSpeedReader" Author="K.A" Created="2002.11.14"
  Desc="EngineSpeedReader">
    <IVICseq>
      <VehicleCommunication>
        <Collection Name="Instruction">
          <Instruction Method="DiagService" Protocol="D2">
            <MethodData ECUAddress="7A" BusType="1">
              <Collection Name="Services">
                <Service Name="A6101D01" Value="A6101D01">
                  <Response ReadOutValue="" />
                </Service>
              </Collection>
            </MethodData>
          </Instruction>
        </Collection>
      </VehicleCommunication>
    </IVICseq>
    <Decoders>
      <Decoding Parameter="RPM" Scaling="40" Max="8000" Min="0" Offset="2" Bytes="2" />
    </Decoders>
  </PowerTool>
</PowerScript>

```

Extract 2 – PowerScript file content

Prototype Application

Figure 22 below shows the main application of the prototype. The main application which is also a FLASH application is a program for illustrating the procedure for localizing a PowerTool to run. This application is by itself not a PowerTool as defined earlier.

The application processing flow is described below. The description addresses different parts of the GUI based on the numbered classification as shown below in Figure 22.

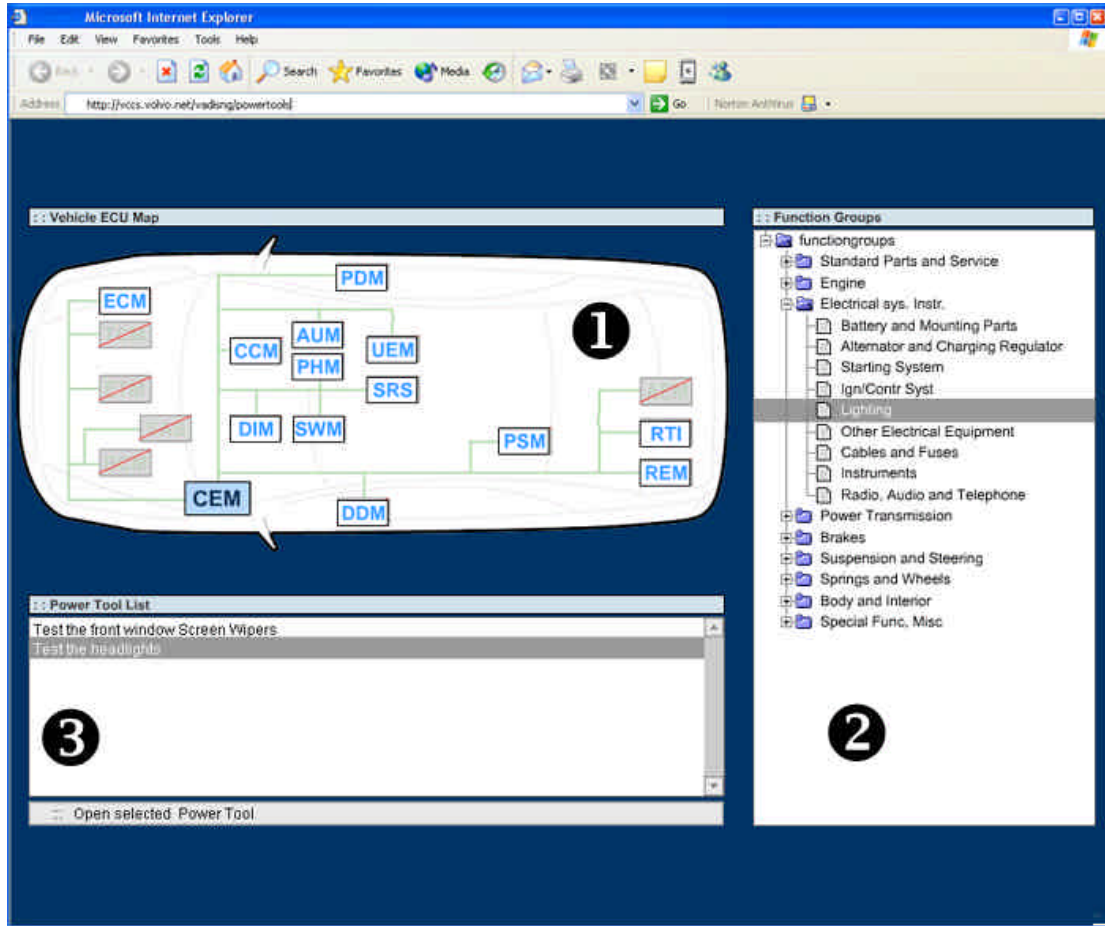


Figure 22 – Snapshot of the prototype application – Main view

- Upon start the available ECU nodes are retrieved and shown in the ECU Map View (part 1). This action is performed by calling the Get Nodes method in vehicle communication web services (CarComWS).
- After the available nodes are retrieved (the state of the app. as shown in the snapshot in) and displayed in the ECU Map, the user can click on any ECU. By doing so the DB containing the PowerTools is contacted and a search based on the chosen ECU nodes ID is done to find PowerTools. This action is performed by calling the GetPTByECU method in the DB communication web services (PowerToolWS).
- The found PowerTool objects are transformed into an XML list and sent back to the application. Based on the XML response a list is created and show in the PowerTools List in the GUI (3)
- When the user click on a certain PowerTool in the list (3) it is marked and also its position in the FunctionGroup tree menu is shown (2). The user can also, instead of

getting the PowerTool list by clicking on the ECU node, navigate in the Function Group tree and find the PowerTool.

- After the chosen PowerTool is selected in the list, the user can click on the “Open Selected PowerTool”-button.
- The PowerTool application is then loaded into the main application. By clicking on the “Open Selected PowerTool”-button the PowerTool FLASH file and PowerScript file are loaded. The path for the files was stored in the PowerTool object retrieved from the DB earlier.

Figure 23 bellow shows a snapshot from the prototype application. It is a case where the user has two PowerTools application up and running (1) (2). The PowerTool in front (2) monitors the vehicle engine speed and stability. The PowerTool (2) GUI consists of a digital and an analog meter along with a “live” graph view. This sample PowerTool (2) is defined by the PowerScript file presented in earlier.

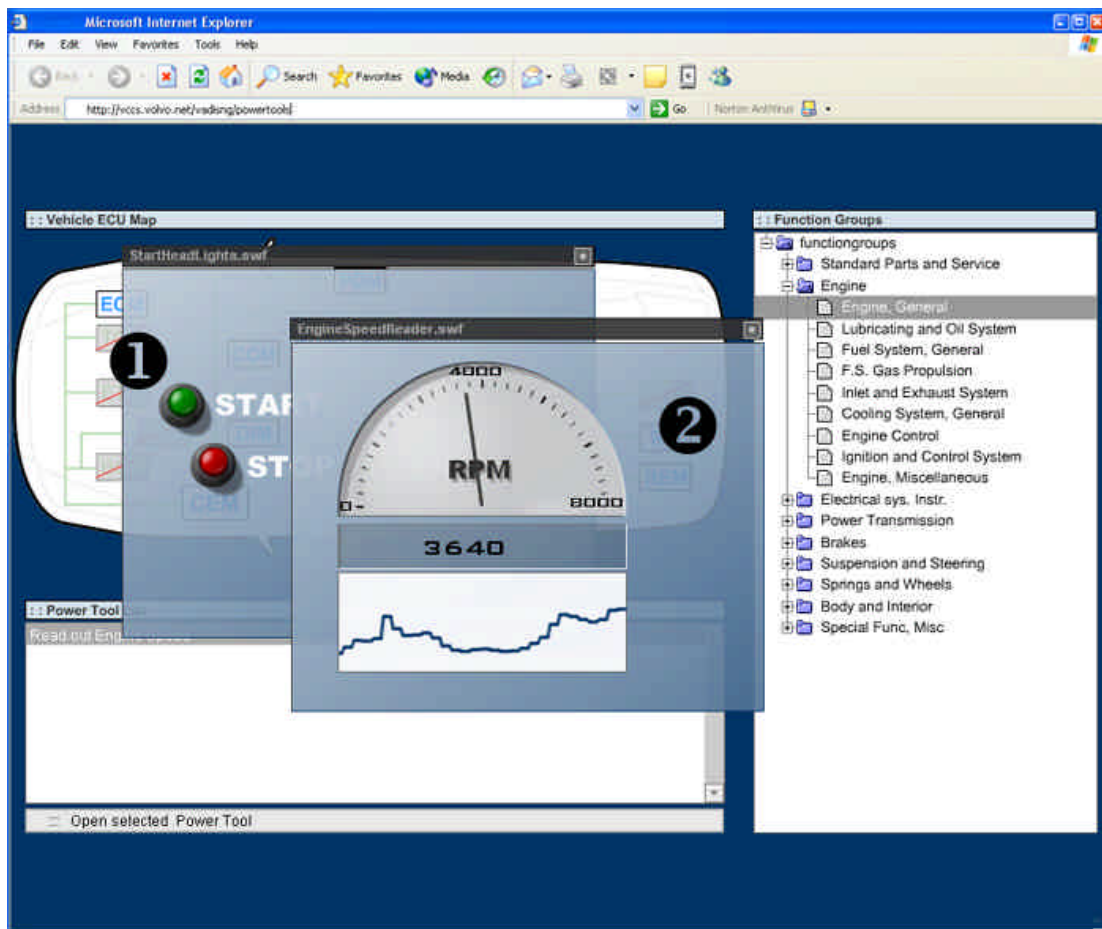


Figure 23 – Snapshot of the prototype, two PowerTools running



Appendix B

- Electronic Computer Unit context model -

Electronic Computer Unit context model

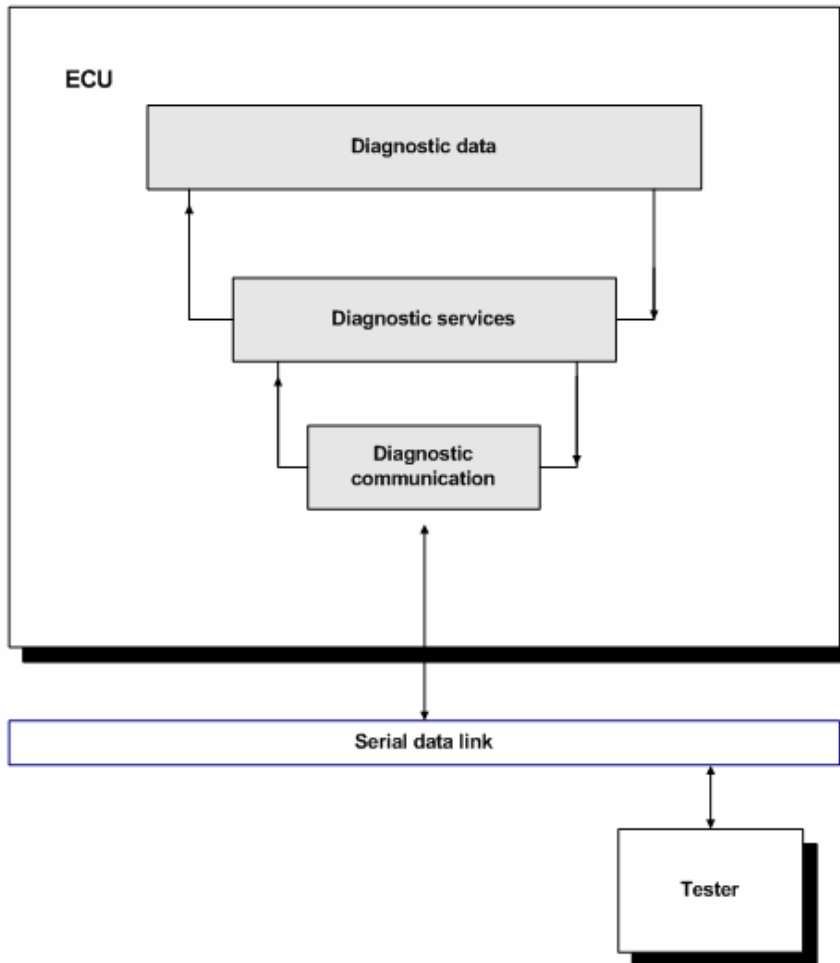


Figure 24 – Context model of ECU and its communication interface

A diagnostic session can briefly be described as follows:

- Tester system sends a request (service request) through the serial link.
- The request is captured from serial link by the ECU
- A service is performed on the ECU and the service response is sent back to the serial link and captured by the Tester.

For better understanding of the concept presented in Figure 24, the core parts of it are further explained as follows;

Diagnostic session

A diagnostic session is started either by a diagnostic initialization (K-Line), or when the ECU has executed a diagnostic service (CAN). A diagnostic session is stopped through either execution of the diagnostic service StopDiagnosticSession, after a time-out, or when the power to the ECU is disconnected.

Diagnostic communication

Exchange of messages between the tester and an ECU. The tester sends request messages upon which the ECU returns response messages. Diagnostic communication is specified in terms of message protocol, structure, timer and error handling.

Diagnostic service

Exchange of information between the tester and an ECU, initiated by the tester in order to read or write diagnostic data or control the behavior of the ECU for diagnostic purposes. Diagnostic services are specified in terms of function and structure and contents of the messages.

Data bytes

In diagnostic messages, data consisting of M bytes is ordered from the most significant data byte in message byte #N to the least significant data byte in message byte #N+M.

Diagnostic data

Data located in the ECU's memory which can be read or written by the tester.

ECU

An ECU - Electronic Control Unit - supports: vehicle control functions; diagnostics defined by Volvo and possibly OBDII or supplier. An OBDII ECU is an ECU that supports both Volvo and OBDII diagnostics.

Tester

An off-vehicle system that controls functions such as testing, inspection, monitoring and diagnostics of an ECU. A tester communicates to ECUs through the serial communication link.

In terms of Volvo Diagnostics II, the prototype application (PowerTools) developed in this project is a so called Tester system. The focus was however not on specific diagnostic service issues, although a deep understanding of the Volvo Diagnostic II concept was required for development of the system.



Appendix C

- Telematics -

Telematics

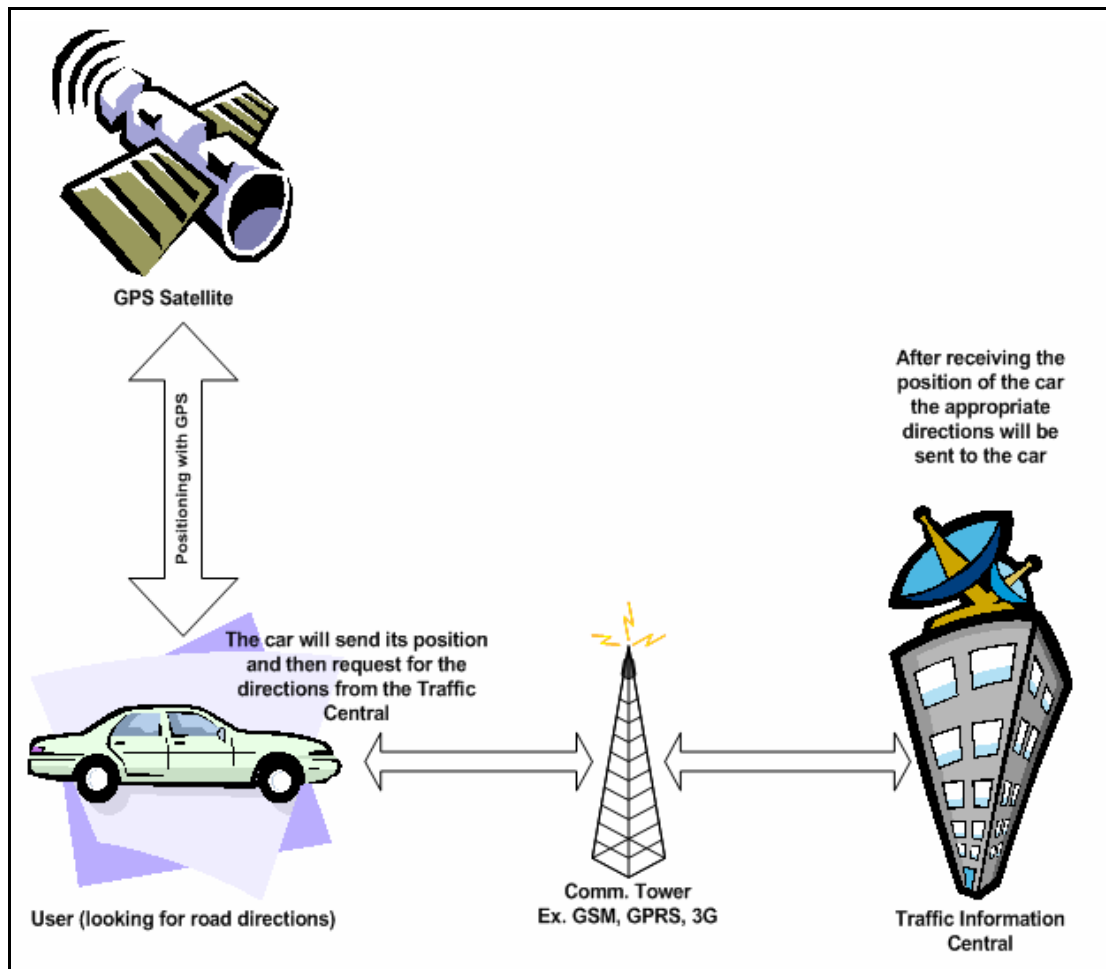


Figure 25 – Example of Telematics system

Figure 25 illustrates an example of Telematics System. Telematics services have like other business services a logical business model. The model is made up of five components ranging from content owners to customers. A model describing the Telematics value chain is presented in Figure 26.

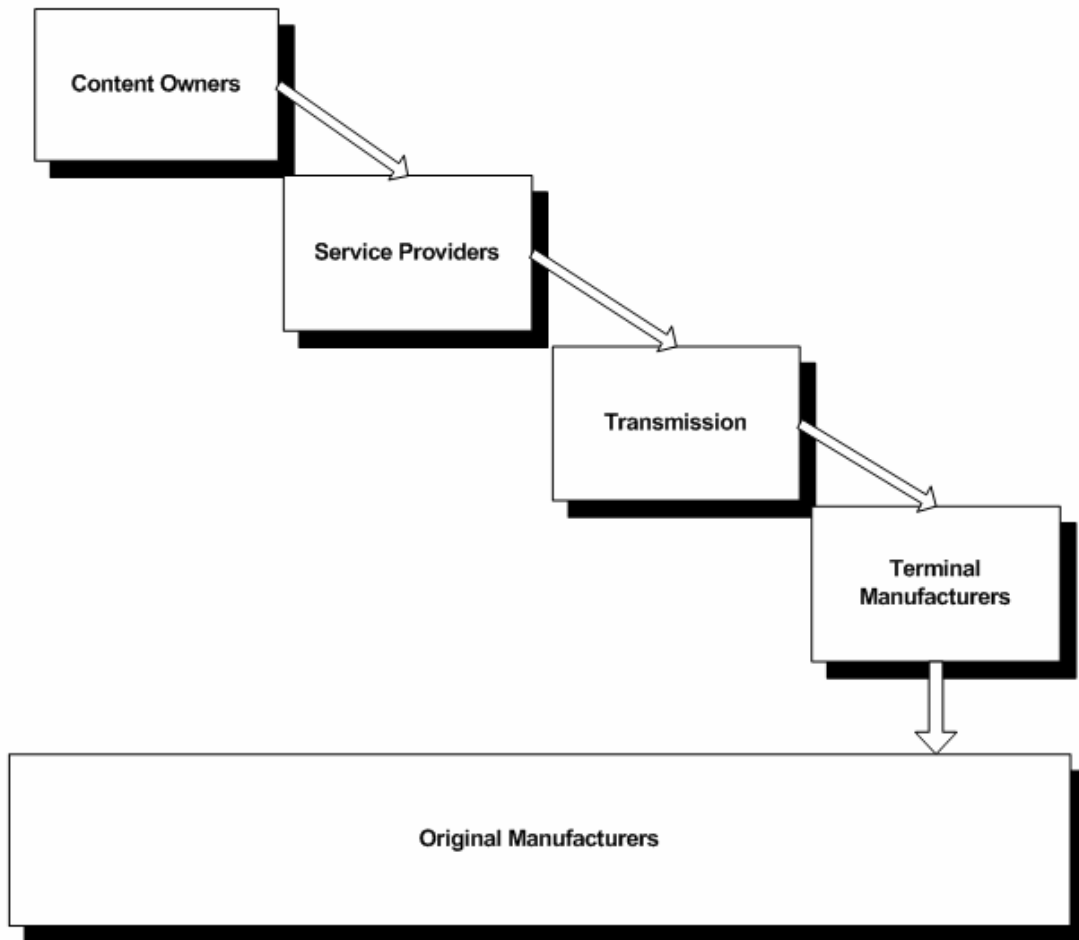


Figure 26 – Telematics value chain

Content owners

Content owners supply the Telematics system with information and data needed. The type of content and the information can vary for different services. Examples of information used by Telematics services are: traffic information, weather forecast, road maps, news and entertainment information. TeleAtlas, Traffic Master, Navtech and Media Mobile are examples of content provider companies. Content owners can also be Volvo (car manufacturer) as they provide service manuals, parts information, etc.

Service Provider / Manager

The service provider has the role of the manager, meaning that this actor has to form the information from the content supplier and make it available for the end-user. The end-user on the other hand does not necessary need to be low level customer but can also possibly be another service provider. Examples of service providers are: OnStar, Wireless Car, Traffic Master and Web site portals like Halebop.

Transmission providers

Transmission providers are the actors that control or own the infrastructure used to transmit the information from the service providers to the end-users. Transmission providers are usually Mobile Network Operators like: Vodafone, Telia and Mannesmann.

Terminal Manufacturers

Terminal manufactures are the type of companies that manufacture Telematics Control Units so called TCUs. TCUs are equipment used in the car to receive and process the transmitted information. Display units with touch screen that are used in the car for interaction with the user for access and use of the Telematics services are example of TCU equipment. Blaupunkt, Autoliv, Sony and Panasonic are among companies that produce such units.

Original Equipment Manufacturers

These actors are the manufacturers of the automobiles. The manufacturers design the systems need to interact with the TCUs. The car owners can be considered as the end-users in the whole chain. Examples of original equipment manufacturers are: BMW, Toyota, Audi and Volvo.