2011

A cloud-based information integration platform for smart cars

Yi Xu
University of Wollongong

SCHOOL OF INFORMATION SYSTEMS & TECHNOLOGY

A Cloud-based Information Integration Platform for Smart Cars

Yi Xu

This thesis is presented as part of the requirement for the
Award of the Degree of
of the
Master of Information Systems and Technology by Research

UNIVERSITY OF WOLLONGONG

August 2011

Certification

I, Yi Xu, declare that this thesis, submitted in fulfilment of requirement for award of Master Information Systems and Technology by Research, in the School of Information Systems & Technology, Faculty of Informatics, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at other academic institution.

Yi Xu

August 23, 2011

Abstract

Although modern Smart Car technologies are growing rapidly, there are many challenges to be addressed. The limited processing capabilities of the contemporary in-car computers in smart cars may cause bottlenecks in smart control. The content in smart car applications can hardly be extended. In addition, the traditional software installation approach that is used in smart cars is neither economical nor convenient. To address these issues in smart cars, we introduced a Cloud-based Information Integration Platform for Smart Cars that has the ability to improve flexibility in the smart car information platform and enhance the value of smart cars. In this work, the architecture of the Cloud-based Information Integration Platform was designed by adapting new technologies, such as Cloud computing, Web Services and Controller Area Network (CAN). This service-oriented solution has a vertically integrated architecture that uses full information gathering and service driven synergies. To achieve the smart control and information sharing in smart cars, the designed platform collects data through the CAN bus automatically, analyses and meshes that data to smart car information services in the cloud. We also designed a Smart Car Information Service as the user interface to implement smart car applications through customized business processes. This fills the information processing gap between the smart car and cloud computing. Furthermore, we can use the information integration platform for different usage scenarios, such as car diagnosis, car repair, etc.

Acknowledgements

I would like to express my gratitude to my supervisor Dr Jun Yan, for his guidance and constant support during my research process. I am also thankful to my friends in Australia for their valuable help. Last but not least, I am grateful to my parents. Without them, I would never be able to have all my achievements.

Publications

1. Yi Xu, Jun Yan. *A Cloud based Information Integration Platform for Smart Cars*. The 7th International Workshop on Mobile Commerce and Healthcare Services (in cooperation with the 2nd International Conference on Security-enriched Urban Computing and Smart Grids), Hualien, Taiwan, September 2011.
2. Yi Xu, Jun Yan. *A Cloud-based Design of Smart Car Information Services*, Journal of Internet Technology, Taiwan, March 2012

Table of Contents

Certification II
Abstract III
Acknowledgements IV
Publications V
Table of Contents VI
List of Figures IX
List of Lists XI
1 Introduction 1
1.1 Background 1
1.2 Problem Statements and Research Motivation 2
1.3 Contributions 4
1.4 Thesis Structure 5
2 Literature Review 7
2.1 SOA 7
2.2 Web Services 10
2.2.1 WSDL 11
2.2.2 SOAP 12
2.2.3 UDDI 13
2.3 Cloud Computing 13
2.4 Software as a Service (SaaS) 16
2.5 Smart Car Composition and Functions 17
2.6 Automotive Electronic Systems and CAN bus in a Smart Car 19
2.6.1 Automotive Electronic Systems 19
2.6.2 Controller Area Network (CAN) 21
2.6.3 CAN Applications 26

2.6.4 OBD II (On-Board Diagnostics II) 27
2.7 In-Car Computer and Wireless Communication 28
2.7.1 GPS and Map Services 29
2.8 Summary 30
3 Cloud-based Information Integration Platform for Smart Cars 32
3.1 Architecture of Cloud-based Information Integration Platform 32
3.1.1 Elements in Designed Platform 35
3.1.2 Human-computer Interface (HCI) in Designed Platform 38
3.2 Message Exchanges and Data Synchronisation to the Cloud 39
3.3 Discussion of Proposed Platform 43
3.4 Summary 44
4 Core Services for the Platform 46
4.1 Smart Car Information Service (SCIS) 46
4.2 CAN Message Service 50
4.3 Location Service 55
4.4 Summary 56
5 Implementation and Simulation 58
5.1 Smart Car Simulation 58
5.2 Implementation of the Smart Car Services in Clouds 62
5.2.1 Using Amazon EC2 to Run Smart Car Information Service 62
5.2.2 Smart Car Services Implementation in the Cloud 67
5.3 Summary 71
6 Case study 72
6.1 Car Repair Facilitation Scenario 72
6.2 Message Flow 77
6.3 Context in the Scenario 79
6.4 Web Service Interfaces Design 84
6.5 Location Data Transformation in Map Service 85
6.6 Summary 91
7 Conclusions 92
7.1 Summary 92
7.2 Discussion 93
7.3 Future Work 95
References 96

List of Appendices 102
Appendix A Orchestration Logic in BPEL 102
Appendix B List of Services Participating in the BPEL Process 103
Appendix C RDFS Code of User Dynamic Context in the Scenario 104
Addendix D RDFS Code of User Static Context in the Scenario 106

List of Figures

Figure 2.1 Basic SOA Services and Messaging Patterns 8
Figure 2.2 Architectural Elements of SOA 9
Figure 2.3 Business Process Triggered by Atomic Web Service 10
Figure 2.4 Business Process and Conversational Web Services 11
Figure 2.5 Representation of Concepts Defined by WSDL 1.1 and WSDL 2.0 Documents 12
Figure 2.6 Car Functions 18
Figure 2.7 One Subset of a Modern Vehicle's Network Architecture 20
Figure 2.8 ISO/OSI Model and CAN Layering 23
Figure 2.9 CAN Physical Connection to the Bus Lines 24
Figure 2.10 Bus Lengths and Bit Rates 24
Figure 2.11 Standard Format Data Frame 26
Figure 2.12 Diagram of Pins on Female On-Board Diagnostics (OBD) Connector 27
Figure 2.13 An Example of an in-Car Computer Interface 28
Figure 2.14 Google Maps 30
Figure 3.1 Overall view of Cloud-based Platform for Smart Cars 33
Figure 3.2 Architecture of Cloud-based Car Information Integrated Platform 34
Figure 3.3 Interface of designed platform 38
Figure 3.4 Cloud Services for Smart Cars 40
Figure 3.5 CAN and IT data bus 41
Figure 3.6 Data conversion process 42
Figure 4.1 The location request and response flow in the platform 55
Figure 5.1 Smart Car Simulation Environment 59
Figure 5.2 Data example in simulation 62

Figure 5.3 AWS Management Window 63
Figure 5.4 Launching and Connecting to Amazon EC2 Instance 64
Figure 5.5 Security Setting of Amazon EC2 Instance 65
Figure 5.6 Using Resources in Amazon Cloud 65
Figure 5.7 IP address for instance 66
Figure 5.8 An Example of CAN message file was stored in the Cloud 68
Figure 6.1 The Business Process of the Car Repair Facilitation Scenario 73
Figure 6.2 CANoe Simulation 75
Figure 6.3 Navigation Service Map 76
Figure 6.4 Message Flow of the Case Study 78
Figure 6.5Application Activity Diagram with Example Context Elements 80
Figure 6.6 RDFS Graph of User Dynamic Context in Scenario 81
Figure 6.7 RDFS Graph of User Static Context in Scenario 83
Figure 6.8 The Class Tree for KML Elements 86
Figure 6.9 Search Result of Designed Service 90

List of Lists

List 2.1 SOAP message - A Sample..... 13
List 2.2 Summary of the Numerous Buses 22
List 4.1Smart Car Information Service Interface..... 49
List 4.2 CAN Message Service Interface..... 52
List 4.3 A CAN Message File..... 54
List 5.1 A message episode from CAN bus under XML format 61
List 6.1 Car Repair Service Interface..... 84
List 6.2 Original Location Database Information from the cloud in a XML File..... 87
List 6.3 Location Information in a KML File 88

Chapter 1

1 INTRODUCTION

1.1 Background

Challenges of urban traffic mainly come from four points:

- energy consumption,
- emissions
- safety
- congestion.

In response to these challenges, the research related to automotive technology has also been changed from the traditional internal combustion engine to electricity and hydrogen for the electrification energy and intelligence research. The characteristic point of green cars is low-emission (zero emission). "Green" and "Smart" are complementary with common development. The cutting-edge technology has just arrived to help us reduce the fuel consumption, resources and time.

The smart car is the future trend in automotive improvement. The development of automotive electronic technology and the technology of smart car is evolving rapidly. The use of new technologies in the smart car has been a focus of study in recent years.

Currently, a smart car mainly uses the Controller Area Network (CAN) bus as the network connection in the body. The CAN bus provides reliable and low-cost connections for automotive Electronic Control Units (ECUs) to support ever-increasing information exchange among them [1]. The data which was collected from ECU's in the smart car, represents the information from our real world. A recent surge of research on CAN in Auto (CiA) has given us new opportunities and challenges. The focus of car design has shifted from an emphasis on transportation to safety and comfort. Car electronics are growing rapidly with the development of low-cost devices. As the performance requirements increase, the demand for in-car computer resources also increases.

In parallel, cloud computing is getting considerable attention, not only from the computer science community but also from the general public. The last two years have seen increased importance placed on research in cloud computing. There has been a dramatic proliferation of research concerned with web services and cloud computing. Cloud computing has been widely used because of its security, sustainability and high flexibility.

1.2 Problem Statements and Research Motivation

On the one hand, the traditional in-car computers have limited processing capabilities to support smart car functions, such as navigation and display of real time information. It is not smart enough without information sharing among the real world and the car computers. With the growth of people's needs in informatics area, the current in-car computers could not meet the demand. On the other hand, the smart car technology has the potential to:

- make the car easier to control
- be more dynamic and economical, and
- be safer to drive.

Currently, the development of smart car applications adopts the same methodology as the development of traditional software. Smart car applications need to be pre-installed in the in-car computer and operate with the local data only. This represents significant limitation because the smart car applications are supported in a limited hardware environment and information sharing is hard to achieve among smart car applications. We advocate that the cloud computing is able to complement the smart cars technologies because cloud computing has many advanced benefits to minimize the hardware and maximize the service. To achieve these benefits, in-car information systems need to be seamlessly integrated with the external information services, supporting the transfer, exchange, and sharing of information among them.

Little research has been done on topic development in cloud-based services for smart cars. In other words, little is known about the conceptual framework and working methods of smart cars' applications. Only few isolated efforts have continued to address the single PC applications for smart cars. While considerable attention has been paid in the past to research issues related to GPS and other car PC functions, literature on issues of services for smart cars has emerged very slowly [2]. The use of cloud computing in the car as a field of research has had limited exploration.

The aim of this research is to design a new information integration platform for smart cars to enhance the ability of processing the real-time information from the smart cars. This enhanced ability will allow for the development and deployment of diverse smart car applications with little effort, thus offering the smart car drivers benefits of safety and an enjoyable driving experience. In this thesis, we present a Cloud-based Information Integration Platform for Smart Cars as a new technology that enables cloud services to seamlessly integrate with the

smart car environment. By adopting the Cloud-based Information Integration Platform, drivers are expected to be able to use their favorite smart car applications and services anytime, anywhere.

1.3 Contributions

Our Cloud-based Information Integration Platform for Smart Cars builds on the results of previous work on CAN applications, in-car computers and cloud computing. In this thesis, we focus on the novel use of CAN bus applications and cloud computing technologies. Our primary contribution is the synthesis of ideas, some of them are novel and unique in their respective areas. It forms the core of a smart car integrated information system.

Some contributions were made to the field of smart cars technology as follows:

- The designed Cloud-based Information Integration Platform for Smart Cars seamlessly integrates the in-car information system with the external cloud information services. With this platform, a designed smart car information service can be easily extended by integrating various services to support new use case scenarios. The smart car users' interest can be communicated via the smart car information service as the interface to smart car applications in the cloud.

- A data conversion approach was developed that is able to convert the data from binary code in CAN bus to the XML format so that they can be processed by CAN message service in the cloud. As a result, we can move many tasks from in-car computers to the cloud. Therefore, the cloud computing plays a more important role in smart cars than the traditional in-car computers.

- The information integration platform for smart cars allows the smart car services to be deployed in the cloud computing environment. It works across multiple smart car models and manufacturers. We demonstrate our approach by and applying our designed platform in a case study to present the technique details, such as the message flow, context of smart car service and the service interface.

- In addition, the implementation of the designed platform presents a novel approach to applying the concept of Platform as a Service. The computing resource to be supplied to smart cars becomes more flexible in the cloud-based platform.

1.4 Thesis Structure

This thesis reports our research on development of the information integration platform in the context of smart cars. The remainder of the thesis is organised as follows:

Chapter 2 of this thesis reviews major technologies that are related to our research. The literature review of the existing knowledge and technologies are discussed, including both cloud computing and the basis of smart cars. In particular, the present study examined the roles of cloud computing, web services and smart cars in the system. This is followed by some background information on the ongoing research within which the present study was carried out and a statement of the specific research issues in smart cars technology.

In Chapter 3, we describe the architecture of the Cloud-based Information Integration Platform for Smart Cars in detail. In this thesis we present a conceptual framework of cloud-based services for smart cars.

We proceed to explain some core smart car services to achieve our designed platform in Chapter 4. We also describe the methodology and procedures of the

data conversion in the cloud-based service platform for smart cars. In order to understand the architecture, we implement the designed cloud-based platform for smart cars in Amazon EC2.

The details of the prototyping are given in Chapter 5. We use CANoe to simulate CAN bus message in smart cars and implemented the core services for the proposed platform.

A car repair scenario is studied as a sample application is shown in Chapter 6 for the purpose of proof-of-concept. It demonstrates the applicability of our approach.

Finally, Chapter 7 concludes this thesis by summarizing the key contributions of this research, discussing the major limitations of the approach and outlining our future work.

Chapter 2

2 LITERATURE REVIEW

2.1 SOA

SOA (Service-oriented architecture) is an architectural style for building enterprise solutions based on services. The real value of SOA lies in its ability to combine services to create dynamic, agile and flexible business processes [3]. From the view of information flow in the service-oriented architecture, SOA includes three entities, i.e.

- service provider
- service requestor and
- service agent.

Generally, Service providers publish various services to a service agent while service requestors find and choose the services they need through the service agent [4]. Based on their characteristics (publisher service, subscriber service and broker service), SOA can be generally described as using a triangle as shown in Figure

2.1. As defined in SOA, a publisher registers its service descriptions with the broker service and a subscriber can discover the services it is searching for if those services are registered with a broker. The broker uses its service registry to recognize matched service descriptions. The subscriber can then invoke a publisher and obtain the requested service [5, 6]. Sometimes the publisher and subscriber services are also referred to as providers and requesters. The communication protocols in these services are supported with standards such as Web Service Definition Language (WSDL), Universal Description, Discovery and Integration (UDDI), and Simple Object Access Protocol (SOAP).

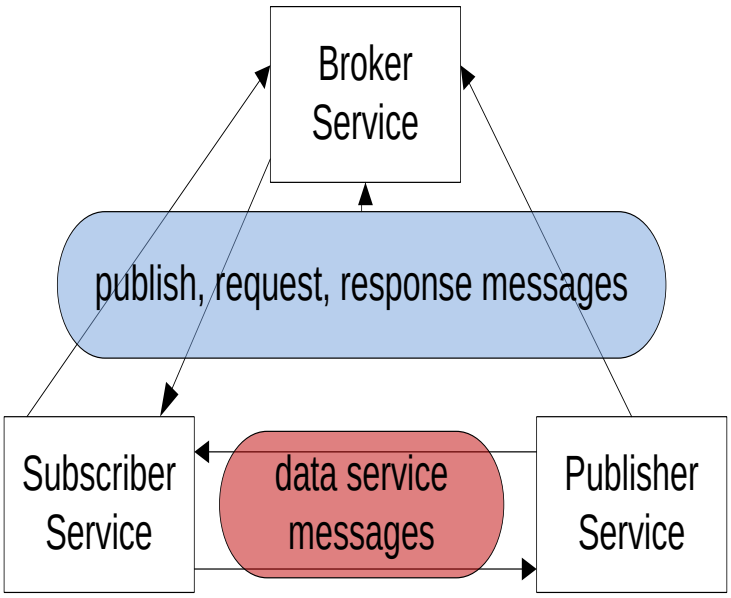


Figure 2.1 Basic SOA Services and Messaging Patterns

As shown in Figure 2.2, the SOA architecture can be represented in four (4) layers:

- integration services layer
- business services layer
- business processes layer and
- enterprise resources layer

Two important concepts are associated with this layer architecture. On the left are the functional concepts that you use to construct systems and processes. On the right are the informational concepts that you use to pass, describe, or manipulate data at those different functional levels [3].

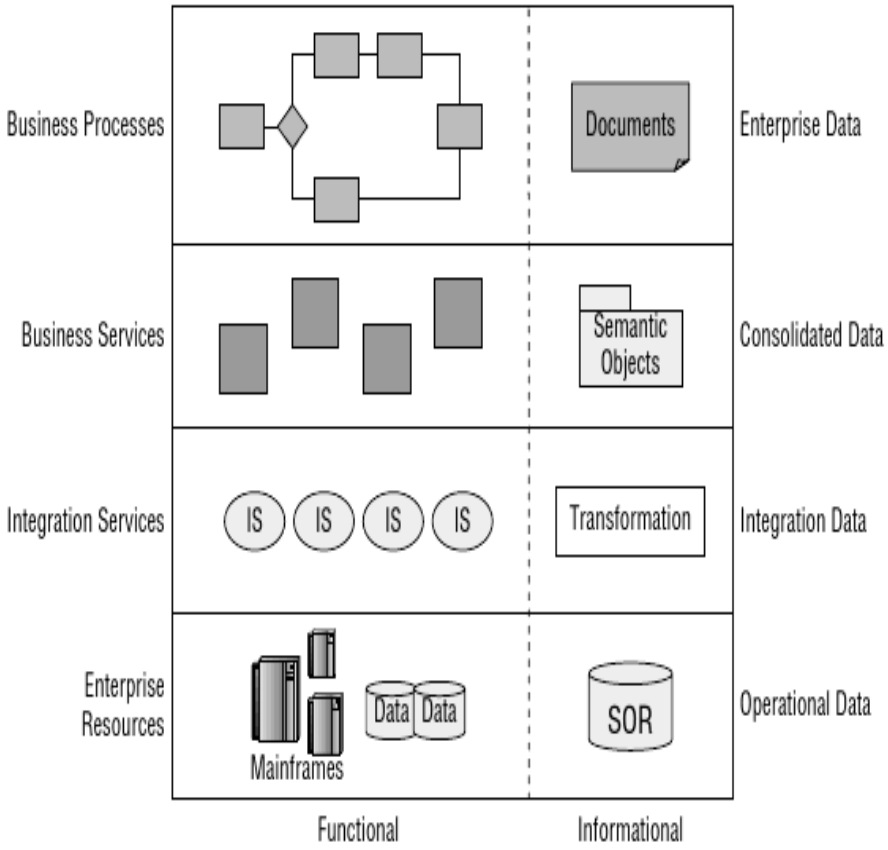


Figure 2.2 Architectural Elements of SOA [3]

The basic concept in SOA enables the creation of services that can be automatically collected to transport desired system dynamics while satisfying several QoS attributes [5, 6].

2.2 Web Services

The W3C defines a "Web service" as "a software system designed to support inter-operable machine-to-machine interaction over a network.

There are four standards that are core to the definition of a Web service and its interface [7-12]. These four core standards are:

- Web Service Description Language (WSDL)
- Universal Description Discovery and Integration (UDDI)
- Extensible Markup Language (XML)
- Simple Object Access Protocol (SOAP)

There are two types of Web services - atomic and conversational. The atomic web service is a kind of single execution of a business process which runs until completion. It is well understood and widely implemented. Figure 2.3 shows that the input message in atomic services contains the complete set of parameters necessary for the business process to run end-to-end [13].

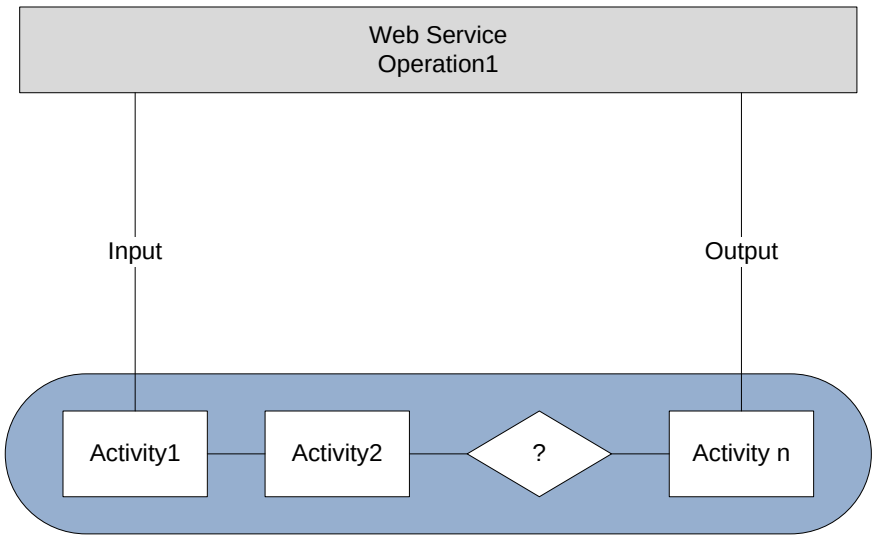


Figure 2.3 Business Process Triggered by Atomic Web Service

By adopting the atomic Web services, the business process in the web services represents the common invocation pattern in traditional e-business architectures. As the result, the atomic interface could directly provide the complete set of the business process input parameters. On the other hand, some more complex business processes dictate that the full set of process parameters can only be developed during the process [13]. This type of communication is shown in Figure 2. 4.

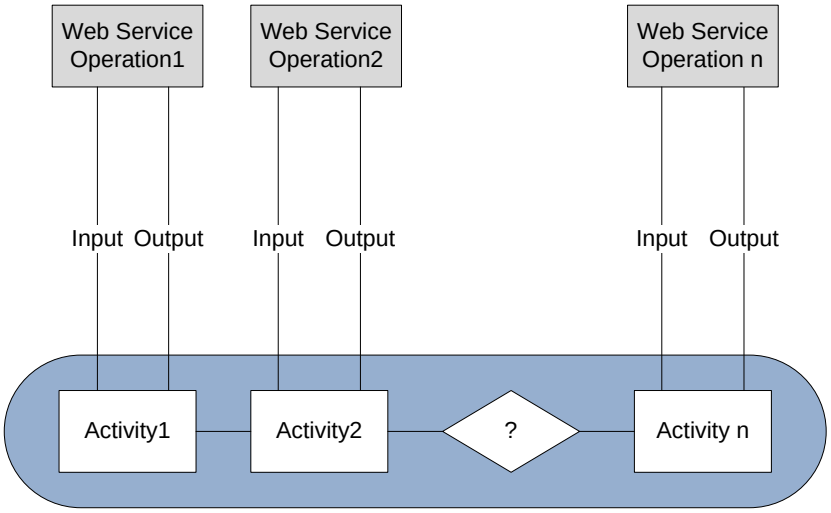


Figure 2.4 Business Process and Conversational Web Services

2.2.1 WSDL

The Web Services Description Language (WSDL) an XML-based language describes the interface of a service including the name and location of the service in SOA [8]. WSDL is the combination with SOAP and an XML Schema to provide Web services. We can use SOAP to call one of those operations listed in the WSDL file [12].

A WSDL document is simply a set of definitions. Figure 2.5 shows the representation of concepts defined by WSDL 1.1 and WSDL 2.0 documents.

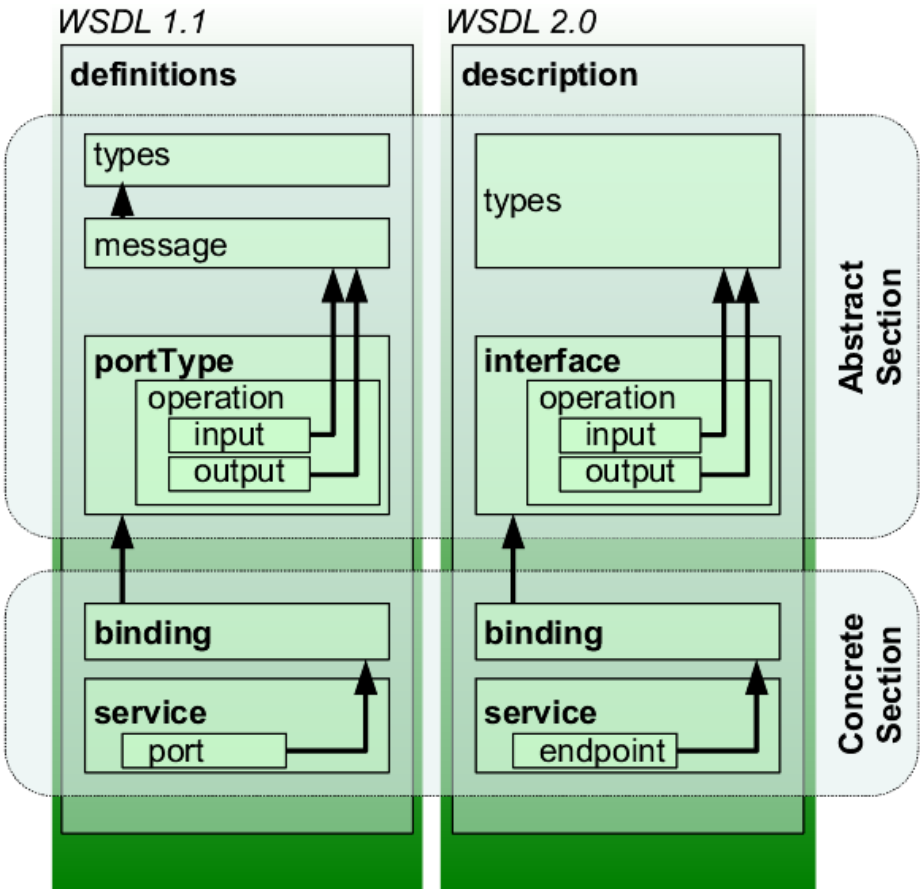


Figure 2.5 Representation of Concepts Defined by WSDL 1.1 and WSDL 2.0 Documents[14]

2.2.2 SOAP

In SOA, SOAP as a XML-based protocol is used for transporting XML messages between a requester and a provider of a Web Service, and UDDI is used for storing information about Web service offerings [8]. The client can call one of the

operations listed in the WSDL file with SOAP [15]. A sample SOAP message is shown in List 2.1 below.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

List 2.1 SOAP message - A Sample

2.2.3 UDDI

Universal Description Discovery and Integration (UDDI) is a core Web service standard. It can be interrogated by SOAP and to provide access to WSDL documents. The provider publishes the WSDL to UDDI and the requester can join to it using SOAP [12]. Using UDDI Registry, in addition to WSDL and SOAP, could reduce integration and development time for legacy enterprise applications [8].

2.3 Cloud Computing

Recent years have seen increased attention being given to the Cloud Computing. Cloud Computing is getting considerable attention not only from academia and

industry but also from people in general public. Cloud computing is a type of distributed computing technology. According to the National Institute of Standards and Technology, cloud computing is defined as "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [16, 17].

Generally, cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services [17, 18]. The datacenter hardware and software is what we will call a "Cloud". By using "Clouds", network service providers can deal with billions or even tens of billions of information in a few seconds, and reach the same powerful performance of a single super computer [17]. For example, some real projects deal with complex combination of working, communication, storage and others. Projects may require the processing of large datasets and use the power of tens, hundreds, or even thousands of computers in the cloud.

Cloud computing is an revolutionary computing model because it makes the services in the Internet flow more freely and become easily accessible. Enterprises and individual users no longer need to purchase expensive hardware. Instead, they only need to hire computing resources in the cloud and pay for the usage. Nowdays, hardware is being developed very fast, the Internet information is growing at an extremely high speed. Using high-speed Internet transmission capacity, cloud computing will process data from a personal computer or server to a large computer cluster. These computers are very common industry-standard servers, readily available from a large processing center that manages them. The center allocates computing resources based on the needs from customers. This will achieve the same effect as a super computer. For example, Marinelli[19] introduced Hyrax, a mobile cloud computing client which allows mobile devices to use cloud computing platforms.

In the cloud computing environment, the cloud extends SOA capability and web-based services , while levelling technical playing field [20]. The relationship between SOA and cloud computing is that cloud computing has the ability to provide IT resources over the Internet on demand, these resources include storage services, database services, information services, testing services, security services, platform services [16]. Meanwhile, some of the most common SOA service types typically include Web services (those that comply with WSDL, XML Schemas, and SOAP) [7].

Amazon's Elastic Compute Cloud (EC2), Google's App Engine, and Microsoft's Azure all provide administrative consoles to manage your infrastructure's capacity within their hosting facility. Amazon's EC2 service (www.amazon.com/ec2) is a good example of the cloud computing. Amazon Elastic Compute Cloud is a web service that provides re-sizable computer capacity in the cloud, which was designed to make web-scale computing easier for developers [21, 22]. In EC2, we can create new Amazon Machine Images (AMI) which contain the applications, libraries, data and associated configuration settings. We can also select the AMI from a library of globally available AMI's [23]. Amazon EC2 allows us to pay only for capacity that we actually use. It changes the economics of cloud computing.

Amazon EC2 provides developers the tools to build failure-resilient applications and isolates themselves from common failure scenarios [24]. By adopting Amazon EC2, we have many benefits as shown below [25-27]:

- Elastic – Amazon EC2 enables us to increase or decrease capacity within minutes. We can launch one, hundreds or even thousands of server instances simultaneously. Moreover, our applications can automatically scale themselves up and down depending on their needs.
- Completely Controlled – We have complete control of our instances. Those instances can be re-booted remotely using web service APIs. We also have access to console output of our instances.

- Flexible – We have the choice of multiple-instance types, operating systems, and software packages. For example, Linux distributions, Microsoft Windows Server and OpenSolaris.
- Designed for use with other Amazon Web Services – Amazon EC2 works in conjunction with Amazon Simple Storage Service (Amazon S3), Amazon SimpleDB and Amazon Simple Queue Service (Amazon SQS) to provide a complete solution for computing, query processing and storage across a wide range of applications.
- Reliable – Amazon EC2 offers a highly reliable environment where replacement instances can be rapidly and predictably commissioned.
- Secure – Amazon EC2 provides many approaches to securing compute resources.
- Inexpensive – We pay a very low rate for the compute capacity we actually consume.

2.4 Software as a Service (SaaS)

The use of external software on the Internet servers is called Software as a Service (SAAS) [17, 28]. SaaS characteristics include [29, 30]:

- Network-based access to, and management of, commercially available software
- Activities managed from central locations rather than at each customer's site, enabling customers to access applications remotely via the Web
- Application delivery typically closer to a one-to-many model (single instance, multi-tenant architecture) than to a one-to-one model, including architecture, pricing, partnering and management characteristics

- Centralized feature updating, which obviates the need for end-users to download patches and upgrades.
- Frequent integration into a larger network of communicating software—either as part of a mashup or a plugin to a platform as a service

SaaS changed the way that software is delivered to the customers. The availability of data, software, hardware and IT services at anytime and in anywhere provides new opportunities to SaaS providers to offer more reliable, fault-tolerant and more economical SaaS solutions [30].

The advantages of SaaS for both end users and service providers are well understood. Service providers could simplify software installation and maintenance and have centralized control over versioning. End users can have "anytime - anywhere" access to the service, share data and act as a team more easily, and keep their data stored safely in the the cloud. With SaaS, cloud-based services platforms can provide applications to users for use as services on demand, either through a time subscription or a "pay-as-you-go" model. In other words, SaaS is a model of software deployment over the Internet. Instead of purchasing the hardware and software to run an application, customers need only a computer with Internet access to run the software[30, 31].

2.5 Smart Car Composition and Functions

Although a driver generally controls a car using the steering wheel, foot pedals, and stick shift or gear shifter, there are thousand of parts in a car, from large metal sheets in the body, to tiny electronic components which control the engine. Some major parts shown in Figure 2.6 below:

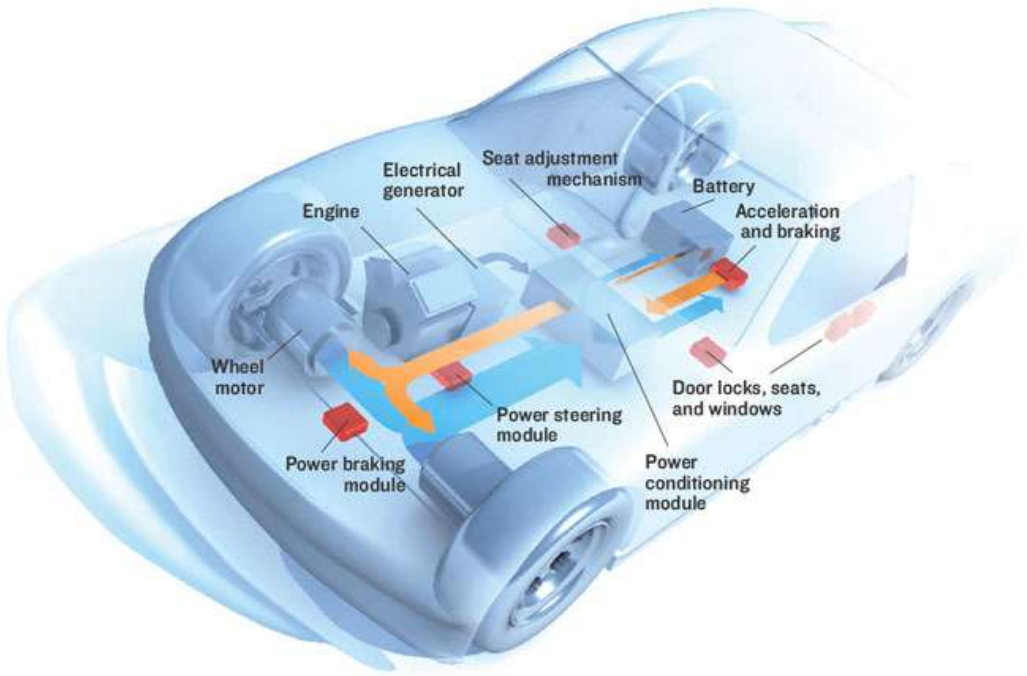


Figure 2.6 Car Functions [32]

We define a smart car as an automobile with intelligence functionality [33, 34]. Currently, smart cars provide a wide range of capabilities. There are many parts inside a modern smart car that are designed to make it comfortable for drivers and passengers. These features include heating air conditioning, electric seats, and music and video systems, safety features include crumple zones, seat belts, air bags, and antilock brakes [35, 36].

The term “Smart Cars Features” refers to many ICT-based systems, including infrastructure systems [37]. Some systems such as anti-lock braking systems (ABS) and electronic stability programme (ESP) systems are already in use. ICT can enable the building of smart cars. These smart systems can help the driver in the driving functions, thus preventing, avoiding or mitigating accidents [37]. Some examples of 'smart' features are as follows [38-48]:

- Adaptive cruise control
- Adaptive headlamps

- Advanced Automatic Collision Notification, such as OnStar [8, 49]
- Automatic Parking
- Automotive night vision with pedestrian detection
- Blind spot monitoring
- Driver Monitoring System
- Vehicle tracking system
- Lane departure warning system
- Pre-crash system
- Traffic sign recognition

2.6 Automotive Electronic Systems and CAN bus in a Smart Car

2.6.1 Automotive Electronic Systems

In a smart car, the automotive electronic system is a very important part for car composition and functions. The car computer is the core of the automotive electronic system as it is the central control of the smart car.

Nowadays, a modern smart car consists of a series of embedded micro-controller systems called Electronic Control Units (ECUs). These units in a car communicate with each other and share information on Controller Area Network (CAN) bus networks. Just as LAN's connect computers, control networks connect a car's electronic equipments. These networks facilitate the sharing of information and resources among the distributed applications [50, 51]. Figure 2.7 shows the sheer number of systems and applications contained in a modern automobile's

network architecture, and the trend toward incorporating ever more extensive electronics.

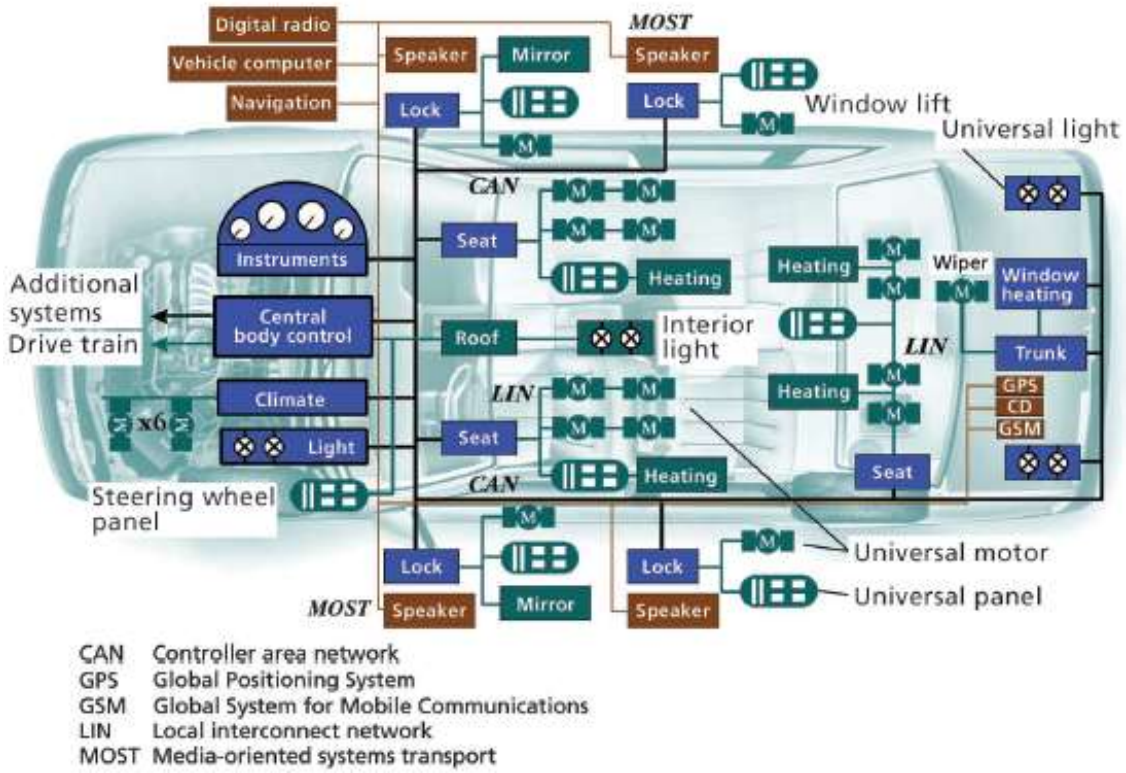


Figure 2.7 One Subset of a Modern Vehicle's Network Architecture [51]

In Figure 2.7, it is easy to find ECU's that can control as follows [52, 53]:

- Engine Management
- Anti-Lock Brake Systems
- Electronic Stability Programs
- "Active" Suspension
- Bodywork Functions

2.6.2 Controller Area Network (CAN)

As we described above, CAN is a serial communication bus designed to provide simple, efficient and robust communications for in-car networks [1, 51, 54] . In a smart car, the car's electronic equipments were connected by CAN bus. It is a serial bus system especially suited for networking "intelligent" devices as well as actuators within a system or sub-system [55, 56].

CAN bus in a smart car provides information sharing platform for all kinds of electronic control systems. Information sharing is the basic function of CAN bus network. Moreover, it is necessary to produce standard control information and achieve information integrated control based on sharing information of different ECUs to enhance the safety, comfort, and added-value of a passenger car and improve its service quality and performance [36].

CAN bus was developed by Robert Bosch GmbH, beginning in 1983, and presented to a wider audience at the Society of Automotive Engineers (SAE) Congress in 1986—effectively the "birth of CAN". In 1987, the first CAN controller chips were released by Intel (82526) and Philips (82C200). In the early 1990s, Bosch submitted the CAN specification (Bosch, 1991) for standardisation, leading to publication of the first ISO standard for CAN (11898) in 1993 (ISO, 1993) [51]. The Controller Area Network (CAN) bus has come to dominate the automotive industry in Europe, and U.S. manufacturers are starting to adopt it [57, 58]. Hundreds of millions of CAN controllers are sold every year and most go into cars. Based on Paret's research [54], we can see approximately 65-67 million vehicles, with 10~15 CAN nodes per vehicle on average. Today almost all of the cars manufactured in Europe are equipped with at least one CAN bus. In the United States, the Environmental Protection Agency has mandated the use of CAN, for On Board Diagnostics, in all cars and light trucks sold in the US from model year 2008 onwards [1, 58]. As a result, CAN was rapidly adopted by the cost-conscious automotive industry, providing an effective solution to the problems posed by increasing vehicle electronics content. Following Mercedes, other

manufacturers including Volvo, Saab, BMW, Volkswagen, Ford, Renault, PSA, Fiat and others, all adopted CAN technology [1].

	LON	CAN	FIP	Profibus	Bitbus	Interbus
General						
Name of protocol	Local operating network	Controller area network	Factory instrumentation protocol	Process field bus	Bitbus	Interbus
Fields of application	Industrial domestic automation	Motor vehicles industrial	Industrial	Industrial	Industrial	Industrial
Original designer	Echelon Corp.	R. Bosch	Schneider	Profibus Consortium	Intel	Phenix contact
Country	USA	Germany	France	Germany	USA	Germany
Users	LONUSERS	CIA	FIP	Profibus Nutzer Organizatin		
Specific characteristics of the physical layers						
Topology	Bus	Bus	Bus	Bus	Bus	Ring
Media supported	Diff. pairs RS 485, CPL infrared	Diff. pairs RS 485, Infrared CPL, Optical fibres	Diff. pairs RS485	RS485	RS485	RS485
Max. length(m)	500	40		1200	120	
For a bit rate (Mb) of	1.25	1				
General characteristics of protocols						
ISO/OSI layers	2 to7	1 and 2	2,7	1,2, 7	1,2,(5) , 7	1,2 ,7
Min. bit rate (kbs)	4.9	1000	31.25	9.6	62.5	30 0
Max. bit rate (kbs)	12500	1000	1000	500	2400	30 0
Max. no. of bytes in message	249(+7)	8(+2)		246(+2)	128(+ 2)	
Priorities	128	2048		2		
Application layers	Lontalk	Numerous	FIP	FMS	RAC	PMS
Electronic components						
References	3150, 3120	a			8*44, 80C 152	IPMS
Manufacturers	Motorola, Toshiba	a		Siemens	Intel	

The list is too long to be shown here

List 2.2 Summary of the Numerous Buses [54]

Based on List 2.2, we can see that the CAN bus offers us a complete solution as a proprietary system. The size of the motor vehicle market (several million units per year) has led component manufacturers to design integrated circuits to handle the CAN protocol. It has also succeeded in reducing costs significantly [57].

As a network technology in a smart car, the CAN bus has its own layers model. Figure 2.8 shows the CAN bus layers and the standard networking model of ISO/OSI [59]. The CAN specification (ISO 11898) has only the physical and data-link layers in a CAN network, and the Data-link layer is subdivided into two parts: logical link control (LLC) and medium access control (MAC) [55, 57].

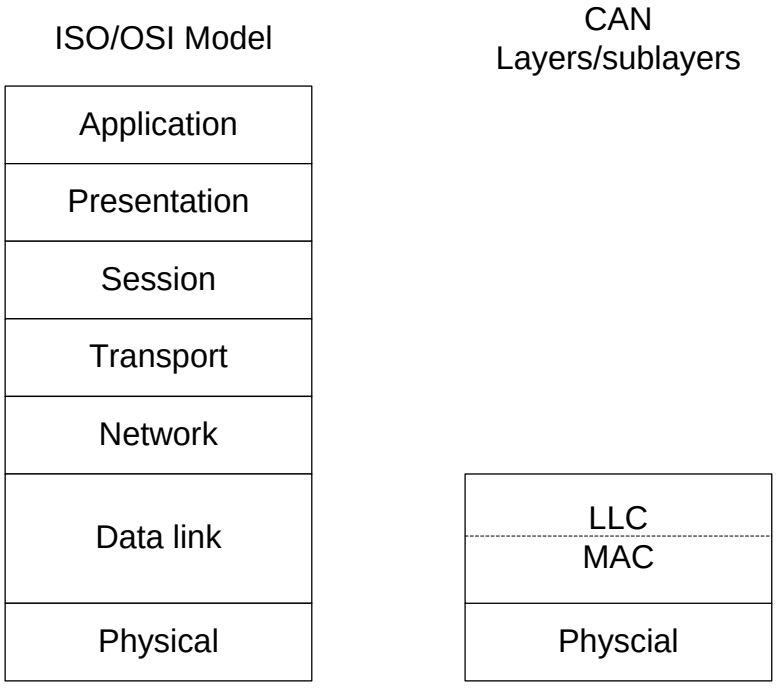


Figure 2.8 ISO/OSI Model and CAN Layering

The function of the physical layer is to transfer the bits from one destination to another, however, it should be noted the physical layer is not a part of Bosch CAN standard. As shown in Figure 2.9, CAN transmits signals on the CAN bus which consists of two wires, a CAN-high and CAN-low. These two wires are operating in differential modes[55, 58]. According the ISO 11898 standard, the impedance of the cable shall be 120 +- 12 Ω, shielded or unshielded [55, 58].

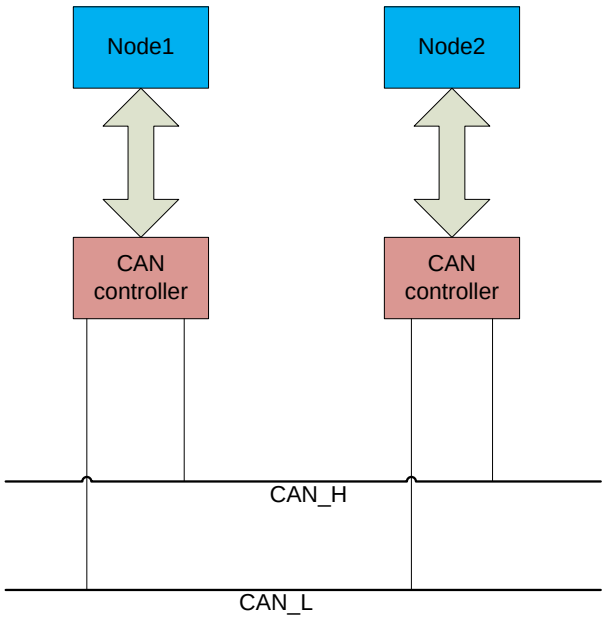


Figure 2.9 CAN Physical Connection to the Bus Lines

Bus length (m)	Maximum bit rate (bit/s)
40	1Mbit/s
100	500kbit/s
200	250kbit/s
500	125kbit/s
6000	10kbit/s

Figure 2.10 Bus Lengths and Bit Rates [55]

In the physical layer, a twisted pair cable is specified with a length ranging from 1,000m at 40Kbps to 40m at 1Mbps [57]. The Maximum bus length for CAN network depends on the bit rate used.

Figure 2.10 shows different bus lengths and corresponding maximum bit rates. Usually, a smart car can contain two or three different CAN's operating at different transmission rates [1, 36, 50, 51]. A higher-speed CAN bus runs more real-time-critical functions such as engine management, antilock brakes, and cruise control. A low-speed CAN runs at less than 125 Kbps, usually and manages a car's "comfort electronics," like seat and window movement controls and other user interfaces. Generally, control applications that are not real-time critical use this low-speed CAN bus which have an energy-saving sleep mode in which nodes stop their oscillators until a CAN message awakens them [1, 36, 50, 51] [30].

Traditionally, CAN bus can link up to 2032 devices (assuming one node with one identifier) on a single network. However, due to the practical limitation of the hardware, it can only link up to 110 nodes on a single network [55].

CAN bus uses Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) to determine access as an asynchronous multi-master serial data bus [1, 58]. Message transfer over CAN is controlled by four (4) different types of frames:

- Data frames,
- Remote Transmit Request (RTR) frames,
- Overload frames and
- Error frames.

The layout of a standard format data frame is shown in Figure 2.11.

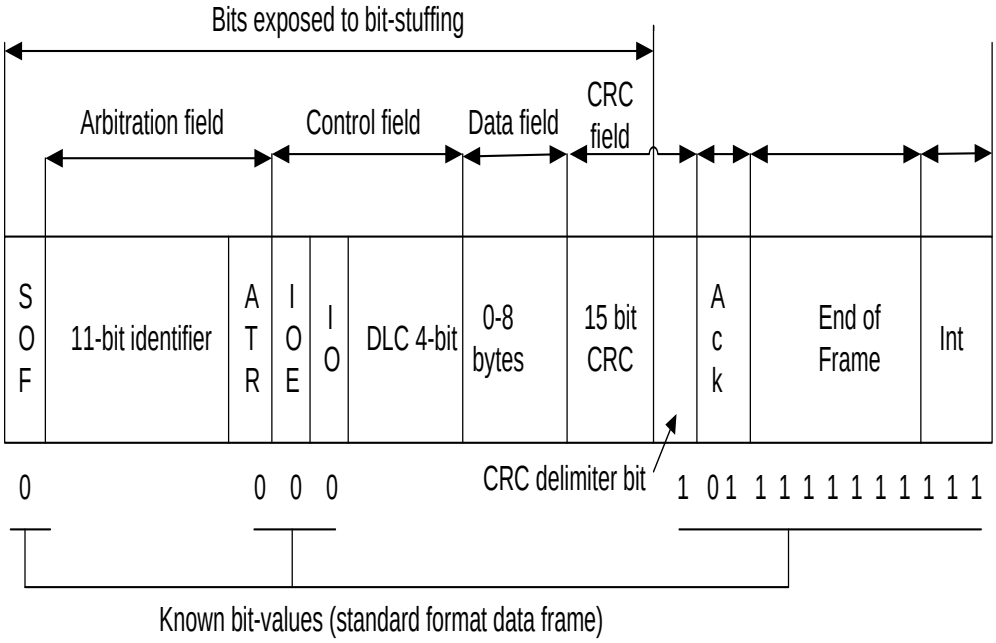


Figure 2.11 Standard Format Data Frame [1, 56-58]

Each CAN data frame is required to have a unique identifier. Identifiers may be 11-bit (standard format) or 29-bit (extended format). The identifier serves two purposes beyond simply identifying the message [1, 56-58]: firstly, the identifier is used as a priority to determine those message contending in the CAN bus; secondly, the identifier may be used by receivers to filter out messages they are not interested in, and reduce the load on the receiver's host microprocessor.

2.6.3 CAN Applications

CAN application layer (CAL) is named and based on an existing protocol originally developed by Philips Medical Systems, CAL is an application-independent application layer that has been specified and is now maintained by the international users and manufacturers group CAN in Automation (CiA) [55].

CANopen is an implementation of CAL and is defined by the CANopen Communications Profile in CiA DS-301 [55]. In CANopen networks, all stations are equal and data exchange is organized directly between devices. Moreover, CANopen can be implemented even on devices with low computing power and storage capacity.

2.6.4 OBD II (On-Board Diagnostics II)

In the early 1990's the automotive industry published the method used for diagnosing vehicle errors electronically, this is embodied in the On-Board Diagnostics (OBD) standard [50]. Originally, the OBD system was implemented to help cars control their emissions; however, the potential of the system was quickly realized and now OBD is the hub of a car's electrical control system [60]. The second generation of the OBD standard, OBD-II, was introduced in 1995 [50]. Figure 2.12 below shows a female On-Board Diagnostics (OBD) connector.

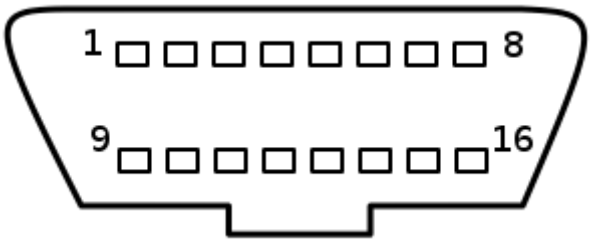


Figure 2.12 Diagram of Pins on Female On-Board Diagnostics (OBD) Connector

2.7 In-Car Computer and Wireless Communication

The in-car computer is the core of a smart car. Traditionally, it is a mini PC or a SCM (Single-Chip Microcomputer) [61, 62]. For example, the CarTel [63] node was described as a kind of in-car computer which runs on the Linux 2.4.31 kernel. Each embedded computer has a high-powered 802.11b miniPCI Wi-Fi card. Another kind of car computer runs Microsoft Windows. E319 [64] is from Prober Industries and does not requires any cables and chargers in a car. The device functions include a 6.5″ touch screen. It can connect to a back view camera so a driver can see what is behind his/her vehicle.. Playing media files is also one of the main functions. The hardware has a 20GB HD with 128 MB of RAM, a 400MHz AMD AU1200 processor as well [65]. The Figure 2.13 shows an interface of the E319.

Figure 2.13 An Example of an in-Car Computer Interface [65]

It is possible to use a car computer to connect to the internet via 3G or 3.5G networks. The computer can be used as a cell phone over GSM networks [64]. In recent years, many 3G and 3.5G wireless networks were deployed in the world, such as CDMA 1x EVolution Data-Only (EVDO), High-Speed Downlink Packet Access (HSDPA), and mobile WiMax [66]. 3G and 3.5G networks have sufficient bandwidth for many smart car applications.

2.7.1 GPS and Map Services

Currently, an in-car computer is commonly equipped with small Global Positioning System (GPS) devices. The Global Positioning System is an earth-orbiting-satellite based navigation system [67]. This system was developed and operated originally by the USA's Department of Defence (DoD). It was designed for, and remains under the control of, the United States military [68, 69]. The fully operational system consists of 21 satellites, with three spares in semi-geo-synchronous circular orbits at a height of 20,200 km, above the earth [69]. For a long time, an in-Car GPS navigation system has been widely used by drivers for multiple purposes, such as discovering gas stations that are within close proximity or providing information about the weather in a location [70]. The navigation system in a car provides route planning, and even voice guidance. By using GPS, users could get their precise position in the worldwide twenty-four hour a day in three dimensions and precise time traceable to global time standards.

Google Maps is a free web mapping service application provided by Google. It powers many map-based services, including the Google Maps website, Google Ride Finder, Google Transit, and maps embedded on third-party websites via the Google Maps API. It offers street maps, a route planner for travelling by foot, car, or public transport and an urban business locator for numerous countries around the world [71]. Depending on the location, we can view basic or custom maps and

local business information, including business locations, contact information, and driving directions [72]. Figure 2.14 shows a basic Google Map.

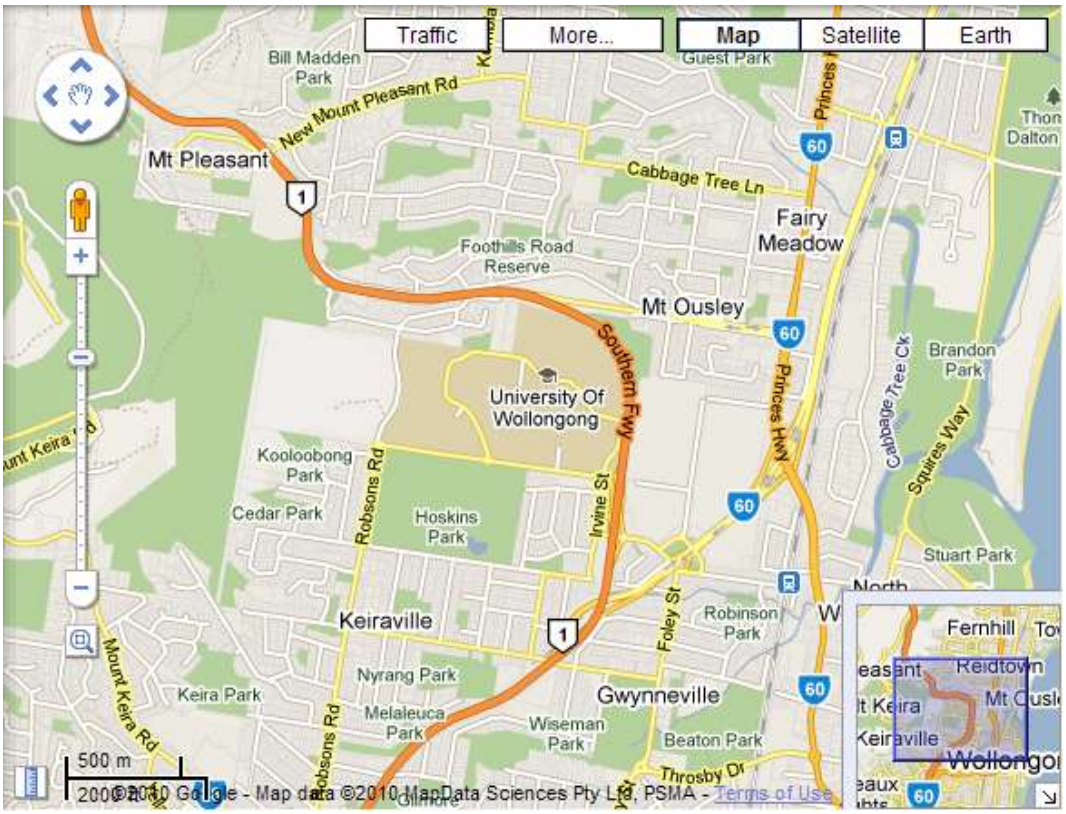


Figure 2.14 Google Maps

2.8 Summary

The importance of smart cars technologies has been recognized and widely accepted by both industry and academic. However, the current in-car computers in smart cars have the limited processing capability, making them bottlenecks. By adopting cloud computing, the content of smart cars can be extended. This gives

the smart car applications the ability to take advantages of the unlimited computing resource in the cloud.

In this chapter we conducted a brief review of cloud computing technologies. SOA as a technology architecture which focuses on building systems based on web services has been extensively studied in the literature since its evolution. In cloud computing, a well designed SOA application will not be limited in computing resources, computing power and processing time. Web Services is widely adopted implementation in SOA. WSDL describes the messaging format of exposed services. Services clients can search the UDDI directories for services and invoke services using SOAP.

We have also reviewed relevant literature about smart cars, such as CAN and CiA. To improve the capability of current smart car applications, we will leverage those smart car technologies with the cloud computing technology.

Chapter 3

3 CLOUD-BASED INFORMATION INTEGRATION PLATFORM FOR SMART CARS

3.1 Architecture of Cloud-based Information Integration Platform

We present the architecture of Cloud-based Information Integration Platform for Smart Cars that enables the delivery of smart car services information. A Cloud-based Information Integration Platform for Smart Cars as a novel and unique platform provides the ability to run many cloud computing applications for smart cars. By using the designed platform, drivers can choose their smart car applications anywhere - anytime. With the help of our proposed approach of exchange CAN bus messages, the ability of the designed platform can be increased because of unlimited smart car services and computing source in the cloud.

We present the novel and unique platform to deal with information sharing from a smart car. As shown in Figure 3.1, all the car applications will be published

as services in the cloud. Remote control, as another access approach for using smart cars applications was provided in this platform.

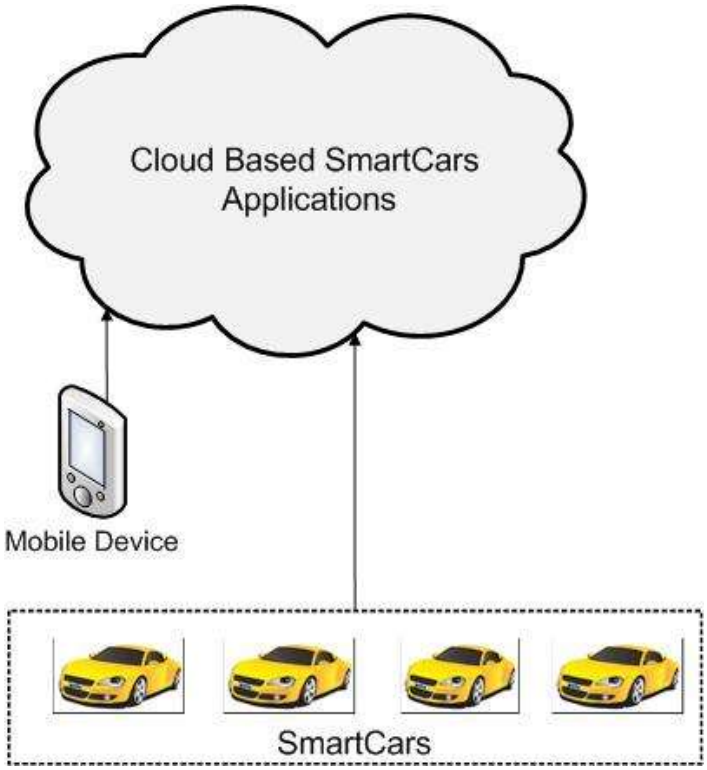


Figure 3.1 Overall view of Cloud-based Platform for Smart Cars

Currently, there are many advanced applications for smart cars we can use, for example, a warning system which will ‘beep’ to remind a driver when unintentionally crossing into another lane;, post-car rear view mirror display system; the driver's blind spot warning system using the side mirror or radar detector to remind him/her that the next car is too near; the automatic parking system and so on. Given the above examples, we propose that all of these applications can now be linked into the cloud computing environment for smart cars.

The Cloud-based Information Integration Platform for Smart Cars has two different layers: namely the hardware layer and the services & software layer. Figure 3.2 shows the architecture of our designed platform.

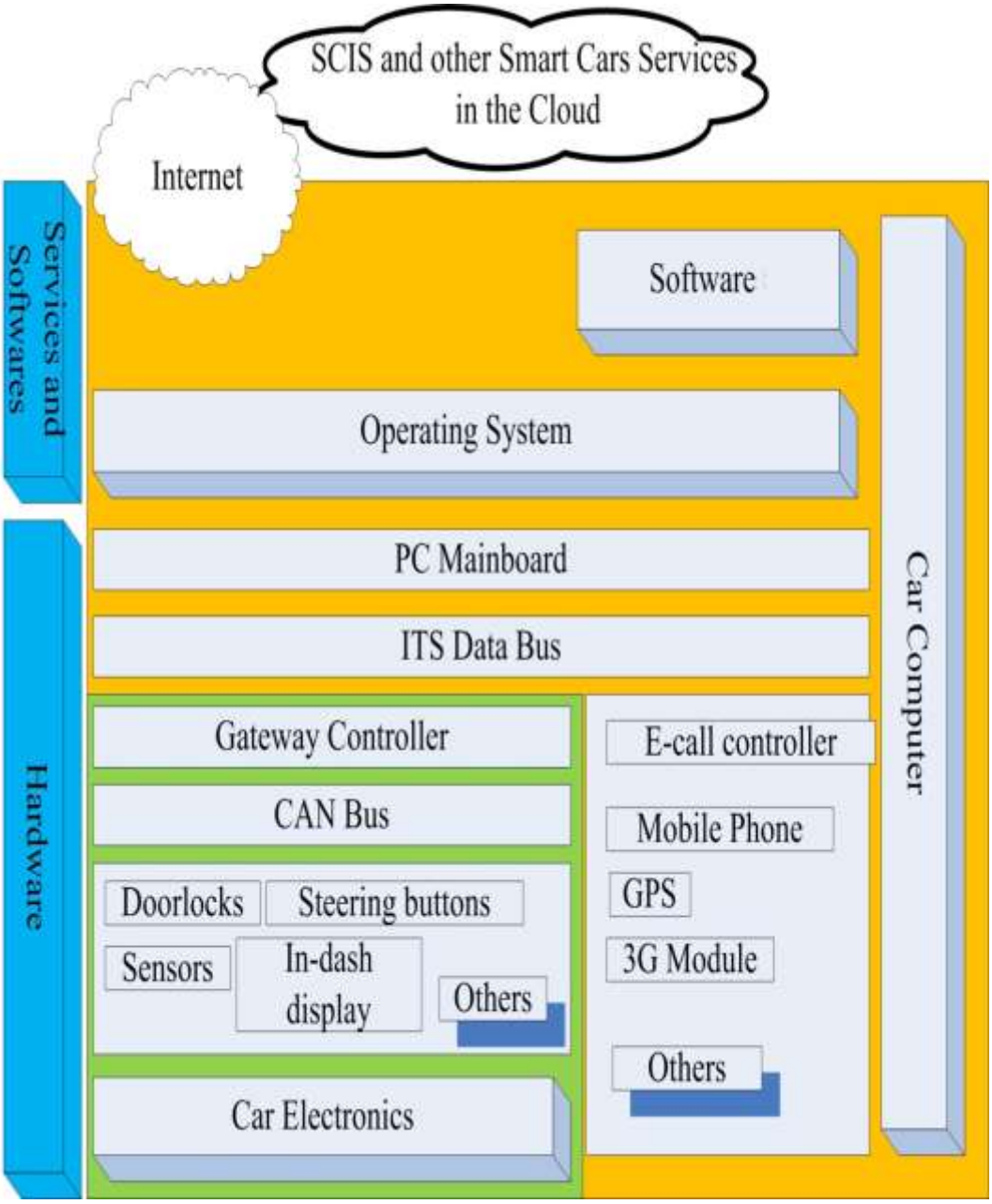


Figure 3.2 Architecture of Cloud-based Car Information Integrated Platform

3.1.1 Elements in Designed Platform

The design of the Cloud-based Information Integration Platform for Smart Cars uses the master-slave structure. The overall distribution of the network structure is a tree. From the perspective of the network topology, the entire communication system is organized by the in-car computer. GPS, wireless, gateway controller, CAN bus and automotive electrical components are controlled by in-car computer in a smart car.

The hardware layer in this architecture acts as the basis of the Cloud-based Information Integration Platform for Smart Cars. This layer includes the CAN bus su-system and an in-car computer. The CAN bus sub-system has frequent internal communication, thus requiring high-demanding real-time control. Most of the car electronics belong to the CAN bus subsystem. In a smart car, an in-car computer will collect messages from sensors via CAN bus. The gateway controller which is between CAN bus and in-car computer can convertCAN bus messages allowing the in-car computer to process the CAN bus messages in the IT data bus. By using the CAN bus, in-car computer and gateway controller, the hardware layer can provide a versatile, low-cost hardware as a base in the information integration platform architecture.

The CAN-bus messages base or extended frame formats will be determined by the use of a higher-layer designed protocol. In our designed platform, the gateway controller provides one procedure to deal with standard CAN bus messages. Normally, different car companies have their own standards, and they may use different format messages in CAN bus. It is possible to process the multifarious CAN bus messages from different companies by using different procedures in the gateway controller.

The gateway controller is positioned between the in-car computer and the CAN bus in a smart car. It is responsible for message conversion between the CAN bus and the IT data bus. The main function of the gateway controller is to analyze all kinds of messages from the CAN bus and convert them into another

format which can be dealt with in the IT data bus. The gateway controller can also issue a directive to CAN, coordinating the work of various ECU's.

Using the gateway controller, messages from each node in the CAN bus can be read or written by the in-car computer. This process is a two-way transmission in CAN bus which provides an information-haring platform for all kinds of electronic control systems to share information [36].

Currently, most cars use two different CAN buses in network connection, a high-speed CAN bus and a low-speed CAN bus. The main mission of the high-speed CAN bus is to connect the engine controller (ECU), ASR and ABS controllers, airbag controllers with those instrument clusters that have the same basic features. The high-speed CAN bus in a car could run more real-time-critical functions to enhance the safety, comfort, and added-value of the passenger car and improve its service quality and performance [36, 51]. For less speed demands, the low-speed CAN bus is used to run a few small wirings that are connected together to form several subsystems. These subsystems use the gateway to share information and coordinate ECUs. The low speed CAN system deals with the connection with central locking, power windows, mirrors and lighting, etc. [54]. The speed of a high-speed CAN bus in a drive system can reach 500kbps, while the speed of the low-speed CAN bus in an electronics system is 100 kpbs [36, 57].

For processing the information in smart cars, the in-car computer will collect information from the gateway controller and deliver those CAN bus messages to the smart car information service (SCIS) in the cloud. It is a bridge between a smart car platform and the cloud-based information service for smart cars.

Global Positioning System (GPS) and wireless communication technology were used in the in-car computer as important roles. The GPS sensor provides the location information of the smart car to a smart car information service (SCIS) in the cloud. The bandwidth of the current wireless network is sufficient for the communication between the smart car and the cloud-based services. The communication device in the in-car computer has the ability of accessing 3.5G and

3.75G networks, can also provide mobile networking access of several Mbit/s to laptop computers and smart phones.

The software and services layer is the higher layer in the architecture of Cloud-based Information Integration Platform for Smart Cars. The designed platform supports the development of various smart car applications in multiple categories through the software and services layer. By using Web Services in the Cloud-based Information Integration Platform, smart car users can choose and use smart car applications as services from the cloud.

The base of the software and services layer is the operation system (OS). Like a traditional computer, current OS applied in the in-car computers include Microsoft Windows, Linux, and Mac OS X. The OS running on the in-car computer manages the hardware and provides common services for smart car applications. In our architecture, the operating system acts as an intermediary between the application programs and the hardware resources.

The software in our architecture supports multiple functions, such as interacting with the cloud-based information services for smart cars and processing sensor data in various formats. For example, the software is able to display the Engine RPM, Coolant Temperature, Fuel System Status, Vehicle Speed, etc.

The Cloud-based Information Integration Platform allows a driver to use smart car applications as services in the cloud. It provides intelligent processing, cloud computing capability and infrastructure to users and various smart car services.

3.1.2 Human-computer Interface (HCI) in Designed Platform

Warning: Low Tyre Pressure (Air slowing out of the tire)

Smart Car Service List:
Car Repair Service
Roadside Assistance Service
Traffice Condition Report Service

Figure 3.3 Interface of designed platform

The goal of using the Human and Computer Interface (HCI) is to make our platform to be more natural. The Figure 3.3 shows the interface of designed platform.

There are a number of interesting HCI devices and areas designed to make it easier to accomplish things with a computer.

We will examine only two (2) in this section. These interfaces have basic applicability in our system. The two (2) areas are touch screen and voice control.

- The touch screen has two main attributes in our designed platform[73]. First, it could enable the user to interact directly with what is displayed, rather than indirectly with a cursor controlled by a mouse or touchpad. Secondly, it allows people to do so without any intermediate device that would need to be held in the hand.
- The car-computer could be controlled by a user's voice. Once removing buttons, dials and switches, car drivers can easily operate appliances without their hands. Speech recognition applications include voice user interfaces such as voice dialling (e.g., "Call home"), demotic appliance control, search (e.g., find a petrol station) and simple data entry (e.g., entering my credit card number) [74].

3.2 Message Exchanges and Data Synchronisation to the Cloud

We present a smart car information service (SCIS) as the interface of the Cloud-based Information Integration Platform for Smart Cars to execute smart car applications in the cloud. Each smart car service has its smart car application backed with different well-defined business processed for smart cars. Smart car services furthers make use of core smart car services that include CAN message service in the cloud. To adopt cloud processing, the SCIS as an access point of cloud-based smart car services and is available for cloud computing in designed platform. In addition, there are many existing technologies in the cloud that can be modified or enhanced to support new smart car applications. As shown in Figure 3.4, one smart car service can act as a service consumer to use the functionality exposed by the other core smart car services. In other words, each cloud-based

service for smart cars can invoke other designed in-car applications by a defined business process in the cloud.

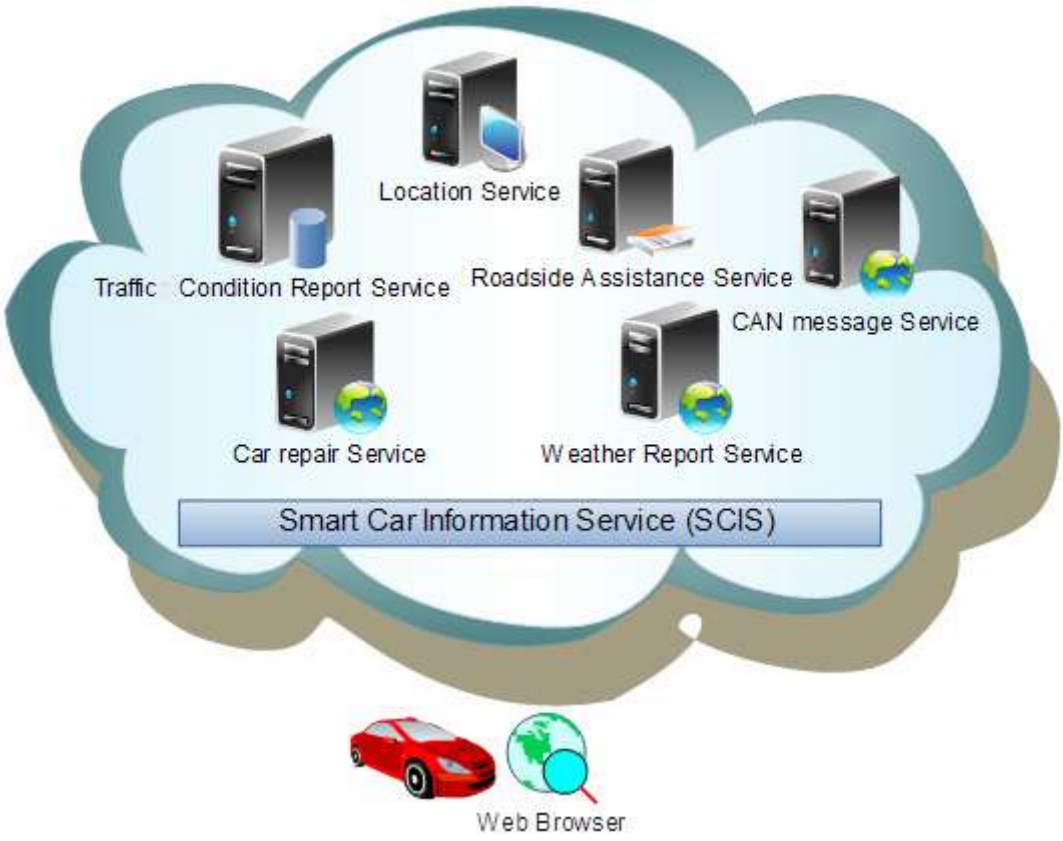


Figure 3.4 Cloud Services for Smart Cars

Due to the principle of the message conversion in the Cloud-based Information Integration Platform for Smart Cars, all the data from the CAN bus will be stored in the cloud via our core smart car service namely CAN message service for analysis and processing. As we described above, the stored data in the cloud can be accessed by other smart car services to develop more various smart car applications.

To achieve the smart car control, we connect the IT data bus and the CAN bus via a gateway controller in the designed platform, the principle as shown in Figure 3.5 below. Currently, the in-car computer uses the IT data bus to exchange

messages among GPS, 3G module, Bluetooth and other in-car devices. The IT data bus requires high-data transfer rate of modules connected together. The gateway controller exchanges messages between IT data bus and CAN bus. In our designed platform, the CAN bus uses the standard protocol with the high data transmission rate to control the smart car electric system, such as door locking, LED displays, steering buttons and so on.

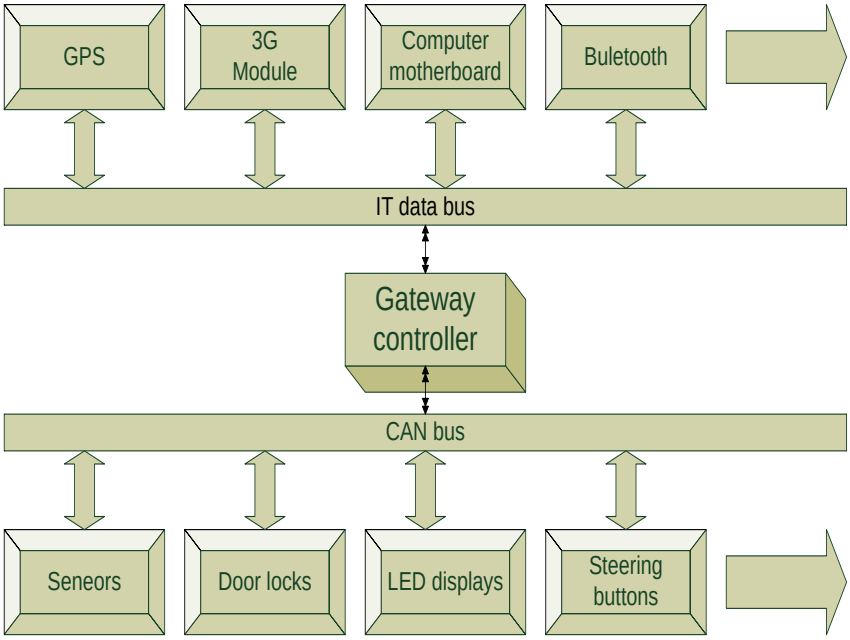


Figure 3.5 CAN and IT data bus

To explore what and how to collect context information from a smart car, we present a CAN message service as the core smart car service in the cloud to process CAN messages which are from a smart car. This service is used for data exchange, applications integration and smart car diagnoses. The CAN message service saves its content data from CAN bus in XML format in the cloud. Access is available through the designed smart car information service as the interface.

Some smart car services in the cloud can periodically update context and data. For example, a traffic condition report service could similarly send to the

navigation service periodic updates of local traffic conditions so that affected users could adapt their travel plans correspondingly.

In our platform, the readily available information attribute from CAN bus includes: Time, Channel, Name, Send node, Data and so on. In a CAN bus registry, the message is stored in the XML format, and data types are defined by using XML Schema. Ideally, we use the CAN message service to deal with the CAN messages XML file. Other smart car services can invoke this CAN message service to archive the smart car information sharing in the cloud. The process is shown in Figure 3.6.

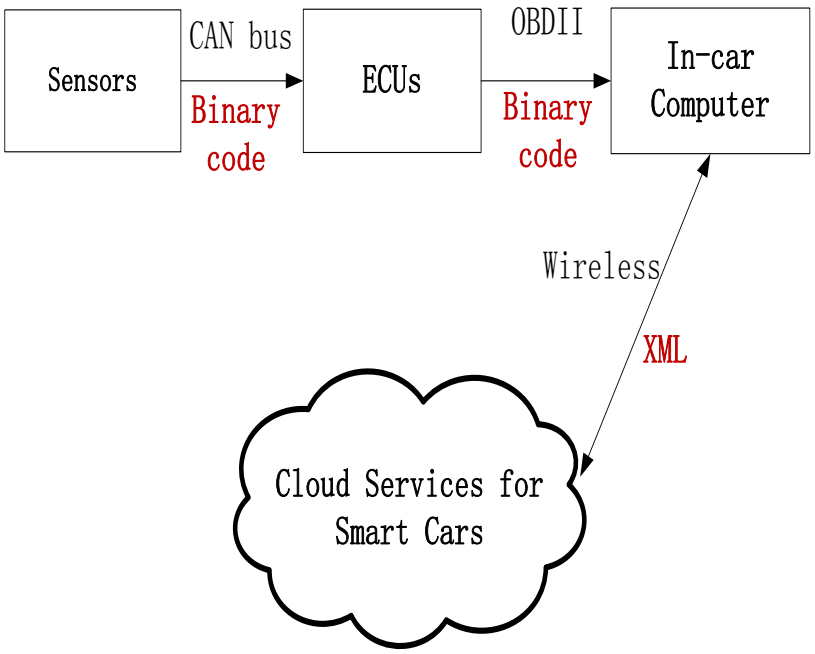


Figure 3.6 Data conversion process

3.3 Discussion of Proposed Platform

The designed Cloud-based Information Integration Platform for Smart Cars is economical and practical for both car users and smart car service providers.

From a hardware provisioning point of view, the new platform does not require high cost hardware for the smart equipment for users. By adopting the cloud computing, most of smart car applications are processed in the cloud. Therefore, our information integration platform can be built with current in-car computer devices with low cost, as the in-car computer designed platform just provides a terminal of those smart car applications.

For information and entertainment providers, using SOA and SaaS, the Cloud-based Information Integration Platform can help smart car service provides to reduce the cost of smart car applications, design, development, deployment and services support. Some smart car functions from a smart car may be turned into cloud services. These changes primarily in software and the operating system level will open up many new areas and add more content to the smart cars. By adopting the information integration platform for smart cars, smart car application developers can design different products in the application layer, while using common hardware standards to achieve flexibility, functionality, low cost and success. Smart car consumers will take control for its functions and achievement.

Cloud processing for smart cars allows car users to start small and increase their budget only when there is an increase in their smart car services or application needs. Further, it provides the possibility for the smart car user to use the internet resource. People could add any features and functionality in a smart car without restriction after it is purchased. In the cloud, infinite service resources are available on demand, quickly enough to enable the provisioning of applications which come from smart car service providers or car companies. Cloud processing for smart car becomes the car's interior and not pre-defined feature which is in the production line. In addition, the cloud-based services for smart cars

are dependable and secure. Users do not need to worry about the problems like safety and privacy while using smart car services in the cloud.

Furthermore, the Cloud-based Information Integration Platform for Smart Cars can possibly help automotive manufacturers and their suppliers follow the rapid development of consumer product technology. Previously, automotive suppliers have found it's very hard to make the smart cars different from others, because of limited available functions. In our proposed platform, the smart car value chain has been improved. By selecting our platform, car manufacturers can reduce their overall investment, but also shorten the learning cycle, improve function and reduce the costs.

3.4 Summary

The current cloud computing and smart car technologies have been extensively discussed in the literature. As the cloud computing has unlimited process resource, the information integrating platform for smart cars is suitable for providing the computing environment to the smart car services.

In this chapter, we adopted the cloud computing to support smart car applications. This aims to bridge the communication gap between the smart cars and vast information services out of the cars. Cloud-based Information Integration Platform allows a driver to use smart car applications as services in the cloud. We presented the architecture of designed platform, and described the elements in the designed platform. By using the gateway controller, information-sharing was provided for all kinds of electronic control systems to share the smart car information. The approach of message exchange and data synchronisation was used in the smart car services. Global Positioning System and wireless communication technology were used in the in-car computer as important roles.

Adopting cloud computing enables smart car applications to be more flexible and intelligent. The Cloud-based Information Integration Platform for Smart Cars

is economical and practical for both car users and smart car service providers. Even more, the automotive manufacturers could get benefits by following the rapid development of consumer product technology.

In the next chapter, we will describe those core smart services in the designed platform, which will support information sharing, integration, and smart control in our designed platform.

Chapter 4

4 CORE SERVICES FOR THE PLATFORM

In this research, we focus on the interaction between the smart car (as the client) and the smart car services (as the server) in the cloud. To support information sharing, integration, and smart control in our designed platform, several core services are designed. These core services include Smart Car Information Service, CAN message service and location service (map service). The details of these services will be discussed below.

4.1 Smart Car Information Service (SCIS)

The proposed information integration platform offers the infrastructural support for smart car services to boost both information sharing and overall performance in smart controls. At the heart of this platform is a core service called the Smart Car Information Service (SCIS). The SCIS is acting as an interface between the smart car users and the smart car services in the cloud. It provides a single-access

point for the user to access various smart car applications. The main functions provided by the smart car information service include:

- User profile management,
- Smart car services registration and discovery, and
- Smart car services access

User profile management is enabled through SCIS and provides an easy, reliable way for managing smart car users' information in the designed platform. User profile management function in SCIS ensures that the user's personal information can be applied to the information integration platform and smart car applications, for the purposes such as user identification, authentication, and authorisation. User authentication is one of the core features in the user profile management that provides SCIS with the ability to identify and control the state of users. This includes the ability to filter users that are currently logged into the Cloud-based Information Integration Platform for Smart Cars, control user login counts and login times. The SCIS will store the personal information to identify the smart car user. Once authenticated, the smart car user will automatically have access to the respective smart car services in the cloud, to which they are entitled. At the same time, SCIS is able to correlate the user with his/her data and information stored in the cloud. Depending on the user case scenario, a user's data and information could include the CAN information of the car that the user is driving, personal configuration information, entertainment data such as music, route points (POI) files, and so on. Therefore, it is easy to synchronise data and services in the cloud when needed. For example, in the cloud-based Information Integration Platform for Smart Cars, users can access their smart car applications from mobile phone or other devices, SCIS ensures users get a consistent experience every time. When users login into the SCIS or launch a smart car application, they will have everything with their own personal settings.

SCIS also allows smart car services to be registered. The registry contains information about the smart car service. Smart car users can discover a needed

service by searching the registry. As we described above, various smart car services are self-contained modules that can be described, published, located, orchestrated, and programmed using XML based technologies. Thus, it is possible to provide a smart car application list or store for smart car user to select their needed services.

As a single access point to users, the SCIS presents to users other smart car services. This is very convenient when a user drives another smart car, as those smart car services he/she already has can be accessed without down-loading the applications. Many SaaS applications for smart cars will be available at little or no cost in the online application market. In addition to lower software costs, IT administration labour costs are reduced as software does not need to be installed and maintained. In the past, software was loaded on many distributed devices, but now, with the use of cloud computing, Smart car services and data can be stored on centralized servers facilitating access to data by a large group of users [17].

The SCIS is designed to be interrogated to provide access to other smart car services. It provides a directory service to smart car users and acts as a proxy for accessing those services. It intends to replace the traditional smart car software, and you can access the SCIS via other devices, such as mobile phones and a table PC. There can be potentially many designed smart car services hosted in the cloud, with well-defined interfaces and in-built business processes. While some smart car services can be atomic services that provide all the functions by themselves, most smart car services will be implemented as compound services by composing multiple existing services in the cloud using a service composition language like BPEL. For example, in our case study presented in Chapter 6, the car repair service is a compound service that uses the functionalities provided by the CAN message service, the station database service and the map service. Using the SCIS, a smart car service is treated as a “black-box”, regardless of its implementation. Smart car users can access this service via the SCIS without the need to be aware of the running process behind its interface. List 4.1 below shows the Smart Car Information Service Interface. According to the SCIS interface, the port "Smart car

information service" defines the operation as “Login”. The login operation requires an input message “LoginRequest". The output message as the response is “LoginResponse”. The user’s information will be sent to the SCIS though the interface to authentication the smart car user.

```
- <definitions targetNamespace="http://175.41.143.46/soap/Smart car information service">
  - <types>
    - <xsd:schema targetNamespace="http://175.41.143.46/soap/Smart car information service">
        <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
        <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/"/>
      </xsd:schema>
    </types>
  - <message name="LoginRequest">
      <part name="UserName" type="xsd:string"/>
      <part name="Password" type="xsd:string"/>
    </message>
  - <message name="LoginResponse">
      <part name="return" type="xsd:string"/>
    </message>
  - <portType name="Smart car information servicePortType">
    - <operation name="Login">
        <documentation>user login.</documentation>
        <input message="tns:LoginRequest"/>
        <output message="tns:LoginResponse"/>
      </operation>
    </portType>
  - <binding name="Smart car information serviceBinding" type="tns:Smart car information servicePortType">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    - <operation name="Login">
        <soap:operation soapAction="urn:sirius_wsdl#Login" style="rpc"/>
      - <input>
          <soap:body use="encoded" namespace="urn:sirius_wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
      - <output>
          <soap:body use="encoded" namespace="urn:sirius_wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
      </operation>
    </binding>
  - <service name="Smart car information service">
    - <port name="Smart car information servicePort" binding="tns:Smart car information serviceBinding">
        <soap:address location="http://175.41.143.46/webservices/login.php"/>
      </port>
    </service>
</definitions>
```

List 4.1Smart Car Information Service Interface

4.2 CAN Message Service

The CAN message service is one of the core smart car services and will be loaded at start up. We designed the CAN message service to collect the CAN data file from smart cars periodically. The CAN data can then be managed as a smart car information resource in our designed platform for various smart car services. The CAN message service can also diagnose the status of the smart car based on the collected CAN data, and notify the smart car user of any abnormal condition via the SCIS. Moreover, the CAN message service is able to identify a smart car. Before processing the CAN message file, the CAN message service will require the user to login to identify the specific CAN message file in its database. By adopting authentication and authorization for accessing CAN message resources, smart car user permissions can be verified before granting access to CAN message service, and user activity can be monitored through various logging records. For user authentication, CAN message service provides a Login() function. One user can have different CAN message database from different smart cars in the cloud.

The cloud-based information integration platform creates the opportunity for making an information-delivery approach from the smart car to the cloud. The CAN message service is important for smart cars to achieve the smart processing with the powerful computing ability of the cloud.

The WSDL document below provides all the information about the CAN message service in an XML document that can be read by a service client.

```
- <definitions targetNamespace="http://175.41.143.46/soap/CAN message service">
  - <types>
    - <xsd:schema targetNamespace="http://175.41.143.46/soap/CAN message service">
        <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
        <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/"/>
      </xsd:schema>
    </types>
  - <message name="LoginRequest">
      <part name="UserName" type="xsd:string"/>
      <part name="Password" type="xsd:string"/>
    </message>
  - <message name="LoginResponse">
      <part name="return" type="xsd:string"/>
    </message>
  - <message name="ImportCANDataRequest">
      <part name="FileName" type="xsd:string"/>
    </message>
  - <message name="ImportCANDataResponse">
      <part name="return" type="xsd:string"/>
    </message>
  - <message name="CANDataDiagnosesRequest">
      <part name="request" type="xsd:string"/>
    </message>
  - <message name="CANDataDiagnosesResponse">
      <part name="return" type="xsd:string"/>
    </message>
  - <portType name="CAN message servicePortType">
    - <operation name="Login">
        <documentation>user login.</documentation>
        <input message="tns:LoginRequest"/>
        <output message="tns:LoginResponse"/>
      </operation>
    - <operation name="ImportCANData">
        <documentation>Upload CAN file</documentation>
        <input message="tns:ImportCANDataRequest"/>
        <output message="tns:ImportCANDataResponse"/>
      </operation>
    - <operation name="CANDataDiagnoses">
        <documentation>Diagnoses</documentation>
        <input message="tns:CANDataDiagnosesRequest"/>
```

```
        <output message="tns:CANDataDiagnosesResponse"/>
      </operation>
    </portType>
  - <binding name="CAN message serviceBinding" type="tns:CAN message servicePortType">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    - <operation name="Login">
        <soap:operation soapAction="urn:sirius_wsdl#Login" style="rpc"/>
      - <input>
          <soap:body use="encoded" namespace="urn:sirius_wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
      - <output>
          <soap:body use="encoded" namespace="urn:sirius_wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
      </operation>
    - <operation name="ImportCANData">
        <soap:operation soapAction="FileName#import" style="rpc"/>
      - <input>
          <soap:body use="encoded" namespace="FileName" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
      - <output>
          <soap:body use="encoded" namespace="FileName" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
      </operation>
    - <operation name="CANDataDiagnoses">
        <soap:operation soapAction="Diagnoses#dia" style="rpc"/>
      - <input>
          <soap:body use="encoded" namespace="Diagnoses" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
      - <output>
          <soap:body use="encoded" namespace="Diagnoses" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
      </operation>
    </binding>
  - <service name="CAN message service">
    - <port name="CAN message servicePort" binding="tns:CAN message serviceBinding">
        <soap:address location="http://175.41.143.46/webservices/CANimport.php"/>
      </port>
    </service>
  </definitions>
```

List 4.2 CAN Message Service Interface

List 4.2 shows the WSDL XML elements of an interface describing a CAN message service. In the example above, the port "Smart car Web services" defines a request-response operation called "ImportCANData". The " ImportCANData" operation requires an input message called " ImportCANDataRequest", and will return an output message called " ImportCANDataResponse" with a parameter called "Filename". A CAN message file can be uploaded to the cloud via the CAN message service interface. As shown in List 4.2, a port type in the CAN message service interface is used for processing specified SOAP messages for diagnoses the CAN message file. For demonstration purposes, we can load the demo CAN message file. This file contains a log of CAN messages. List 4.3 shows our standard OBDII CAN message in a XML file. This CAN message file illustrates how to use XML to store CAN message and signal information from the smart car.

```
<?xml version="1.0" ?>
- <EvoScanDataLogger>
  - <vehicle name="Universal OBDII CAN" LastUpdated="2011">
    - <ecu name="CAN OBDII">
      - <Mode2 name="Mode2" type="Data List Items">
          <DataListItem DataLog="N" Color="" Display="Engine Load"
            LogReference="EngineLoad" RequestID="01" Eval="x" Unit="%" MetricEval=""
            MetricUnit="" ResponseBytes="1" GaugeMin="0" GaugeMax="255" ChartMin="0"
            ChartMax="255" ScalingFactor="1" Notes="(current airflow / peak airflow)
            * (Baro And AirTemp Compensation) * 100%." Priority="1"
            Visible="False" />
          <DataListItem DataLog="N" Color="" Display="Engine Coolant Temperature"
            LogReference="CoolantTemp" RequestID="02" Eval="x*1.8+32" Unit="degF"
            MetricEval="x" MetricUnit="degC" ResponseBytes="1" GaugeMin="0"
            GaugeMax="255" ChartMin="0" ChartMax="255" ScalingFactor="1" Notes=""
            Priority="1" Visible="False" />
          <DataListItem DataLog="N" Color="" Display="Short Term Fuel Trim"
            LogReference="STFT" RequestID="03" Eval="x" Unit="%" MetricEval=""
            MetricUnit="" ResponseBytes="1" GaugeMin="0" GaugeMax="100" ChartMin="0"
            ChartMax="100" ScalingFactor="1" Notes="-ve%(lean) +ve%(rich) Fuel
            trim correction being utilized by the fuel control algorithm, in closed-
            loop fuel control. If the fuel system is in open loop, STFT will report 0%
            correction." Priority="1" Visible="False" />
          <DataListItem DataLog="N" Color="" Display="Short Term Fuel Trim 2"
            LogReference="STFT2" RequestID="04" Eval="x" Unit="%" MetricEval=""
            MetricUnit="" ResponseBytes="1" GaugeMin="0" GaugeMax="100" ChartMin="0"
            ChartMax="100" ScalingFactor="1" Notes="-ve%(lean) +ve%(rich) Fuel
            trim correction being utilized by the fuel control algorithm, in closed-
            loop fuel control. If the fuel system is in open loop, STFT will report 0%
            correction." Priority="1" Visible="False" />
          <DataListItem DataLog="N" Color="" Display="Long Term Fuel Trim"
            LogReference="LTFT" RequestID="05" Eval="x" Unit="%" MetricEval=""
            MetricUnit="" ResponseBytes="1" GaugeMin="0" GaugeMax="100" ChartMin="0"
            ChartMax="100" ScalingFactor="1" Notes="-ve%(lean) +ve%(rich) Fuel
            trim correction being utilized by the fuel control algorithm, in both
            open-loop and closed-loop fuel control." Priority="1" Visible="False" />
          <DataListItem DataLog="N" Color="" Display="Long Term Fuel Trim 2"
            LogReference="LTFT2" RequestID="06" Eval="x" Unit="%" MetricEval=""
            MetricUnit="" ResponseBytes="1" GaugeMin="0" GaugeMax="100" ChartMin="0"
            ChartMax="100" ScalingFactor="1" Notes="-ve%(lean) +ve%(rich) Fuel
            trim correction being utilized by the fuel control algorithm, in both
            open-loop and closed-loop fuel control." Priority="1" Visible="False" />
          <DataListItem DataLog="N" Color="" Display="Short Term Fuel Trim Bank 1"
            LogReference="STFTBank1" RequestID="07" Eval="x" Unit="%" MetricEval=""
            MetricUnit="" ResponseBytes="1" GaugeMin="0" GaugeMax="100" ChartMin="0"
```

List 4.3 A CAN Message File

The advantage of using web service interfaces to deal with CAN messages is that we can receive standard-compliant results from the web services. We can easily integrate these results along with other standard-based mapping components,

into other smart car applications to create more powerful smart car applications for users. As we mentioned before, the CAN message service has the ability to identify different smart cars. Once the CAN message service receives the forwarded message from the SCIS, the CAN message file in the cloud will be linked to the personal account. The CAN message service will process the correct CAN message file in the cloud since the user login the SCIS. The CAN message file contains parameters of the CAN bus under the XML format.

4.3 Location Service

Location is critical information for many smart car services such as roadside assistance and travel planning. Therefore, a location service is one of the core facilitating smart car services. We use the following approach to access the location information. Current 2G and 3G networks use the Location Enabler to provide location data to location applications. 3GPP standards organization has formulated the Open Service Access (OSA) specification that allows third party applications to access network services, including the access to location information [8]. The location enabler supports mobile location services for operators, subscribers and third party service providers [75].

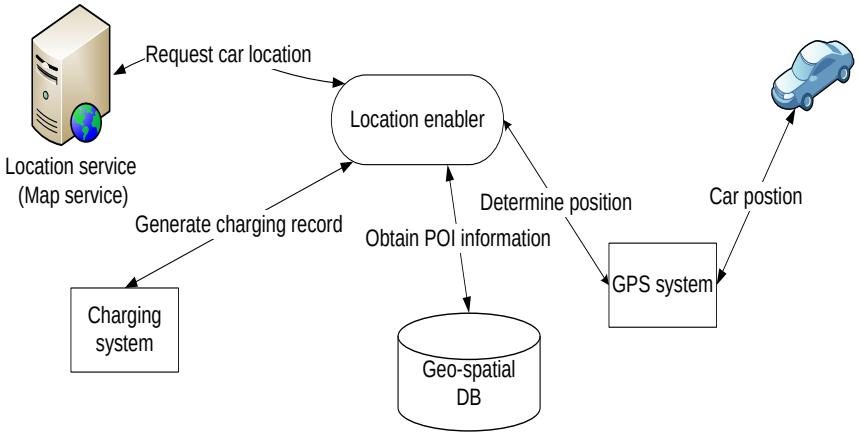


Figure 4.1 The location request and response flow in the platform [8]

The context model diagram in Figure 4.1 illustrates our location request and response flow. The location enabler provides the location information from other coordinate data sources such as charging system, geo-spatial database and GPS system. The location service collects the correct coordinates of the smart car from the location enabler by sending a car location request. Then other smart car services just invoke it to get the location data. The location service is one of the background smart car services in the cloud. It is working to address the gap between the real world information and smart car services, and lends itself to the smart car concept. OMA's [8] location specification defines the core set of operations that a location server that supports a location enabler function should be able to perform. Other smart car services are designed to use the same location service in sync, which is also a natural fit for smart cars.

We choose the Google map service as our location service in our Cloud-based Information Integration Platform for Smart Cars. The Google Maps API is designed to be faster and more applicable to mobile devices, as well as traditional desktop browser applications [76]. The API allows us to create robust map applications in the cloud. Furthermore, we can access the Google Maps over HTTPS, allowing us to utilize the API within the HTTPS secure application.

4.4 Summary

To achieve the interaction between the smart car and the smart car services in the cloud. We introduced core services including the Smart Car Information Service, the CAN message service and the location service (map service) to support information sharing, integration, and smart control in Cloud-based Information Integration Platform. The main functions provided by the smart car information service include user profile management, smart car services registration and discovery, and smart car services access. The CAN message service is designed to process the CAN data file from smart cars. By using the

CAN message service, we create an approach for information delivery from smart cars to the cloud. Besides, the location service as core smart car service can support the location information sharing from the smart car to the cloud-based smart car services. The relationships between the three core services are dynamic. Figure 6.4(Message Flow of the Case Study) shows their particular relationship in our case study.

Chapter 5

5 IMPLEMENTATION AND SIMULATION

To demonstrate the key ideas of this research, we implemented a proof-of-concept prototype. CANoe [52, 77] was used to simulated the CAN bus environment in a smart car. We implemented our project in Amazon EC2. It was selected because it provides all the needed capabilities, in terms of intercepting the loading, modifying the classes and also there are implementations available for cloud computing providers and clients on top of this platform. Moreover, using Amazon EC2 has PAYG (pay as you go) solution to users.

5.1 Smart Car Simulation

Our example demonstrates the functions of the J1939 TestFeatureSet. In the scenario below, an ECU will be tested for tire pressure control. The ECU is simulated in CANoe. CANoe is a comprehensive software tool for the development, testing and analysis of entire ECU networks and individual ECUs [77, 78]. In our research project, CANoe is used to create CAN bus simulation models. Both graphic and text-based evaluation windows are available for us to

evaluate the results. Figure 5.1 shows the smart car CAN bus working in a design scenario in the simulation platform.

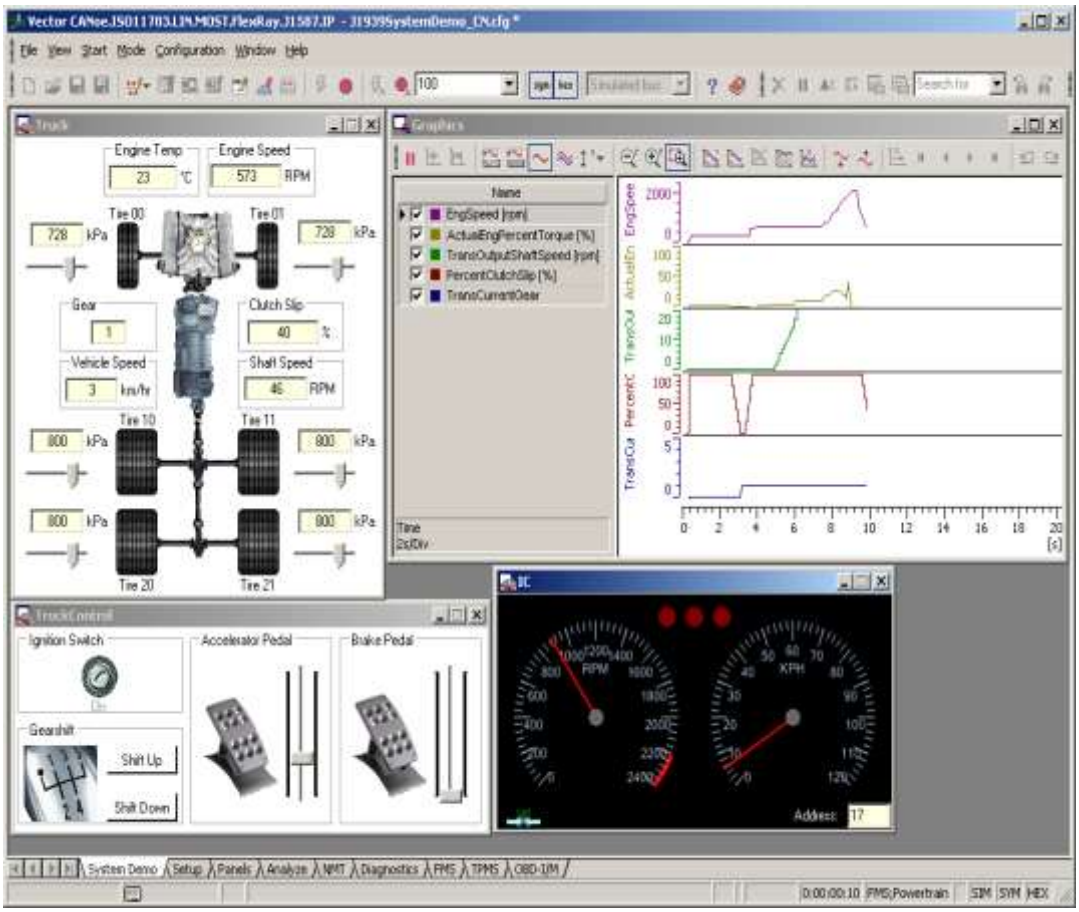


Figure 5.1 Smart Car Simulation Environment

The I/O interfaces of the CAN bus hardware can be simulated completely by CANoe. CANoe environment variables offer us a generic interface for this type of data exchange. We just create an abstraction layer. Those details of using CANoe as follow:

- **Simulation**

In our scenario, all specific functionality of an ECU was simulated by a nodelayer-DLL. We utilize the functionality of this DLL by means of function

calls from a generated CAPL program. CAPL (communication access programming language) from Vector [79] is used as the programming language for building the smart car simulation platform. This language for this scenario is based on C programming language.

- **Creating a test application**

Those simulated ECU's were controlled by a test sequence which we created previously.. It is easier to create a test application in CANoe.

- **Communications monitoring and analysis**

We use the trace window to trace the CAN message traffic. We not only watch the service that is currently being processed, but also see all relevant service parameters at the same time. The CANoe Scanner evaluates CAN messages and shows the active nodes in a list. Other node-specific information is also displayed, such as the node state and device name.

- **Interfaces**

CANoe uses the standardized XML file format (XDD and XDC) which is defined in CiA 311 [77, 78] to store data. For example, List 5.1 below shows a portion of the simulated data generated by CANoe.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?>
<!--   Version 1.0   -->
<!DOCTYPE LoggingExport (View Source for full doctype...)>
- <LoggingExport>
  - <header hexdec="hexadecimal" symnum="symbolic" timemode="absolute">
      <columntitle>Time</columntitle>
      <columntitle>Chn</columntitle>
      <columntitle>PGN</columntitle>
      <columntitle>Name</columntitle>
      <columntitle>Send node</columntitle>
      <columntitle>Src</columntitle>
      <columntitle>Dest</columntitle>
      <columntitle>Prio</columntitle>
      <columntitle>Dir</columntitle>
      <columntitle>DLC</columntitle>
      <columntitle>Data</columntitle>
    </header>
  - <event timestamp="0.000556" bustype="CAN" channel="1" fgColor="#008000" bgColor="#ffffff">
      <col name="Time" />
      <col name="Chn">1</col>
      <col name="---">EE00p</col>
      <col name="Name">ACL</col>
      <col name="Send node">TPMS</col>
      <col name="---">51</col>
      <col name="---">all</col>
      <col name="---">6</col>
      <col name="Dir">Tx</col>
      <col name="DLC">8</col>
      <col name="Data">4F 34 A9 E8 00 26 00 00</col>
    </event>
  - <event timestamp="0.260576" bustype="CAN" channel="1" fgColor="#0000ff" bgColor="#ffffff">
```

List 5.1 A message episode from CAN bus under XML format

In this list, the “Data” field in the CAN bus message was augmented with some attributes. For example, in our case study, the second part of the message represents the pressure of the tyre. Suppose we receive a data string from a normal situation:

```
Data 10 C8 60 24 FF FF FF FF
```

The highlighted value C8 indicates that the type pressure of 800kPa is at the normal level. Figure 5.2 shows the data collected from the CAN bus in the

simulation platform. In that time point (second 70.261140), the tyre pressure is below the normal. B6 means only 728kPa in the tyre. According to the CAN bus simulation, we know that the status of the tyre is abnormal and the tyre was damaged, however, it can be repaired.. The warning message from the CAN message service will let the driver know that the faulty tyre should be replaced soon.

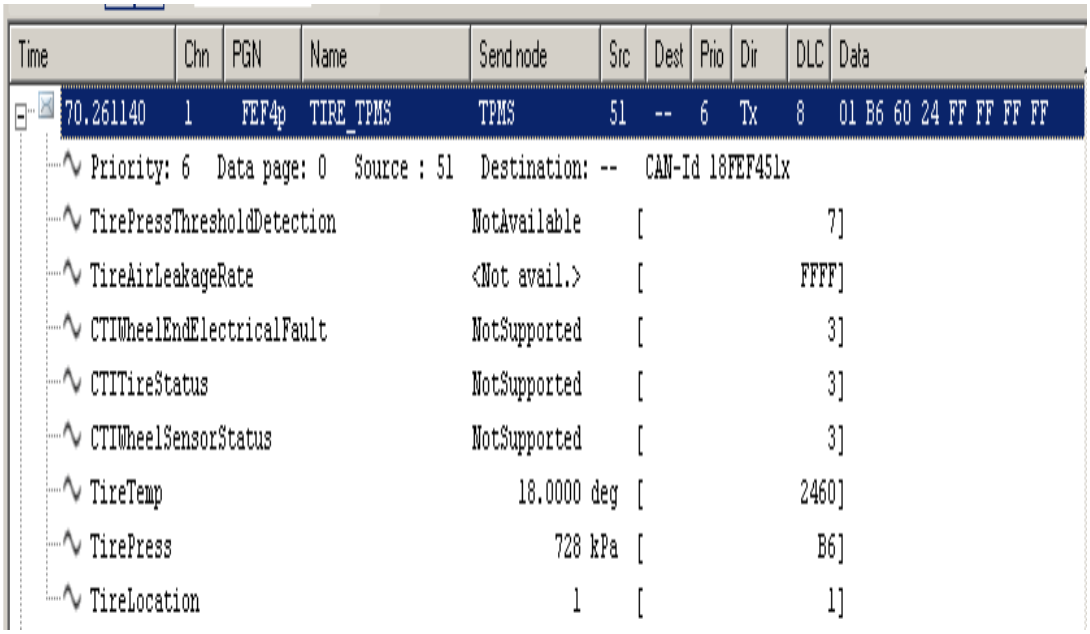


Figure 5.2 Data example in simulation

5.2 Implementation of the Smart Car Services in Clouds

5.2.1 Using Amazon EC2 to Run Smart Car Information Service

We use Amazon EC2 to run our designed smart car services. Amazon EC2 allows us to do scalable deployment of applications by providing a web service through which we can boot an Amazon Machine Image to create a virtual machine. This

virtual machine is named an "instance" and contains preinstalled software [21]. We can create, launch, and terminate server instances when needed, paying for the usage of active servers only. Even more, we can set up several instances in different regions in the worldwide, making one back up the other.

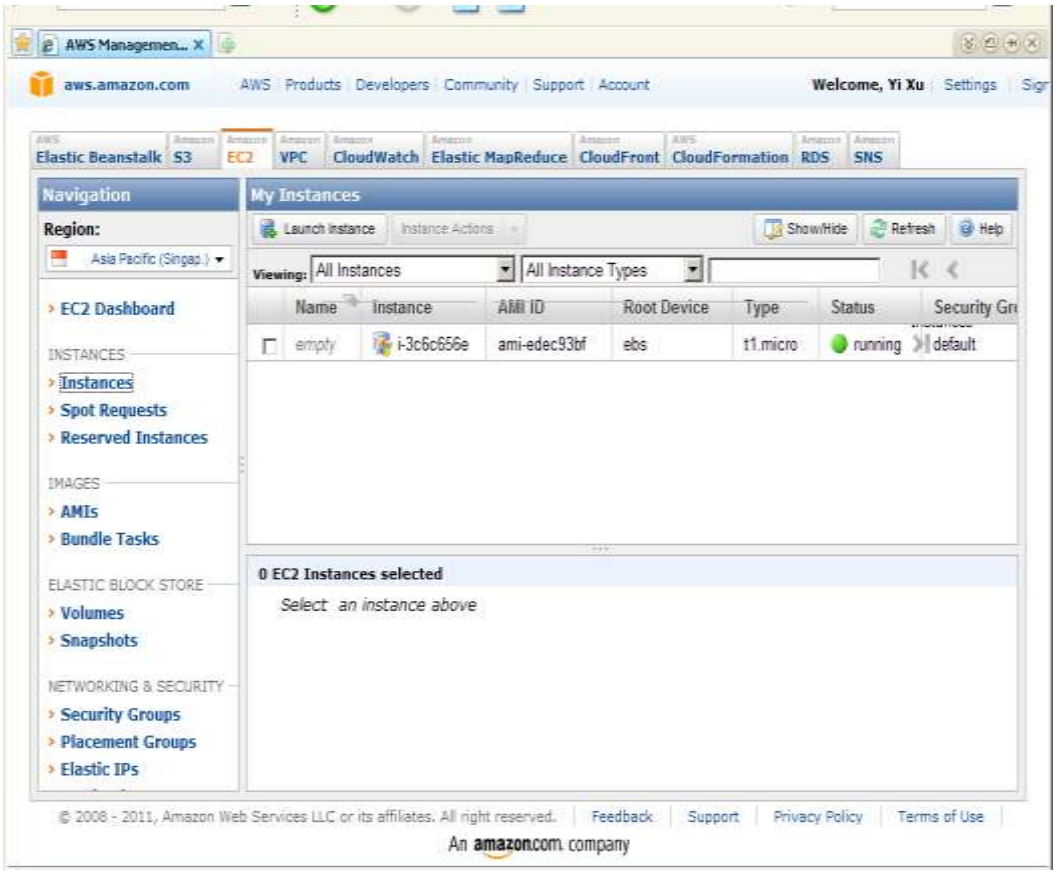


Figure 5.3 AWS Management Window

Steps for using Amazon EC2 to implement the Smart Car Services are shown below:

1. We used the AWS Management Console to create an AMI in Amazon EC2. The AWS management window is shown in Figure 5.3. We selected a pre-configured window2008 image to get up and running. This Amazon Machine Image (AMI) contains applications, libraries, data, and associated

configuration settings that we need, such as IIS. Once we have set up our own account and uploaded AMIs, we are ready to boot the instance. The AMI is simply a packaged-up environment that already includes all the necessary bits to set up and boot our instance. It is a unit of deployment. Therefore, we can compose our system out of several building block AMI's (e.g., webservers, appservers, and databases). We can start the AMI on any number and any type of instances by calling the *RunInstances* API [24]. In our example, we just want a simple window server, we choose one of the standard window distribution AMI's. Figure 5.4 shows the process of launching and connecting to Amazon EC2 instance. Amazon EC2 allows us to set up and configure the instance from the operating system to applications.

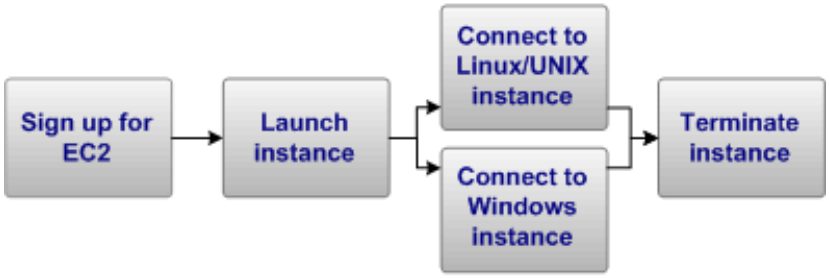


Figure 5.4 Launching and Connecting to Amazon EC2 Instance

2. The setting of security and network access configuration on our Amazon EC2 instance is shown in Figure 5.5. Access can be given on a port-by-port basis, or for a whole network. For example, to identify a web service, use port 80,if the network port is available, all client requests will be forwarded directly to the server.

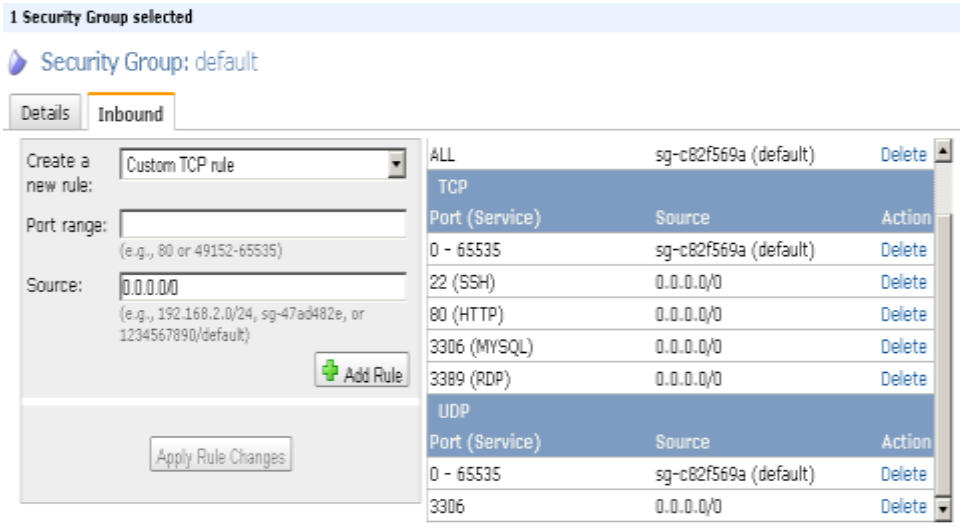


Figure 5.5 Security Setting of Amazon EC2 Instance

3. We use the web service API's and the management tool provided by Amazon to monitor those instances and other resources. Figure 5.6 shows our current using resources in Amazon Clouds. Amazon EC2 web service allows us to obtain and configure capacity with minimal friction. It provides us the complete control of computing resources in Cloud computing environment.

Figure 5.6 Using Resources in Amazon Cloud

4. We utilize static IP endpoint to our instance. Our IP address is 175.41.143.46, which is provided by Amazon. The IP address window is shown in Figure 5.7.

Figure 5.7 IP address for instance

We only need to pay for the resources that we actually consume. Amazon EC2 changes the economics of computing by allowing payment for capacity that we actually use, like instance-hours or data transfer.

For the purpose of simplicity, in this research, we use the Smart Car Information Integration Platform to deliver smart car applications across the Web for use as a service on demand, through a “pay-as-you-go” model. The Smart Car Information Integration Platform uses SaaS in the cloud, i.e. those smart car services were hosted in Amazon EC2 and the users are accessing them via Web. As the software in the Amazon EC2 are already licensed for a single user or for a group of users, the users do not need to spend more money on infrastructure, thus avoiding expensive, time-consuming and laborious upgrade processes.

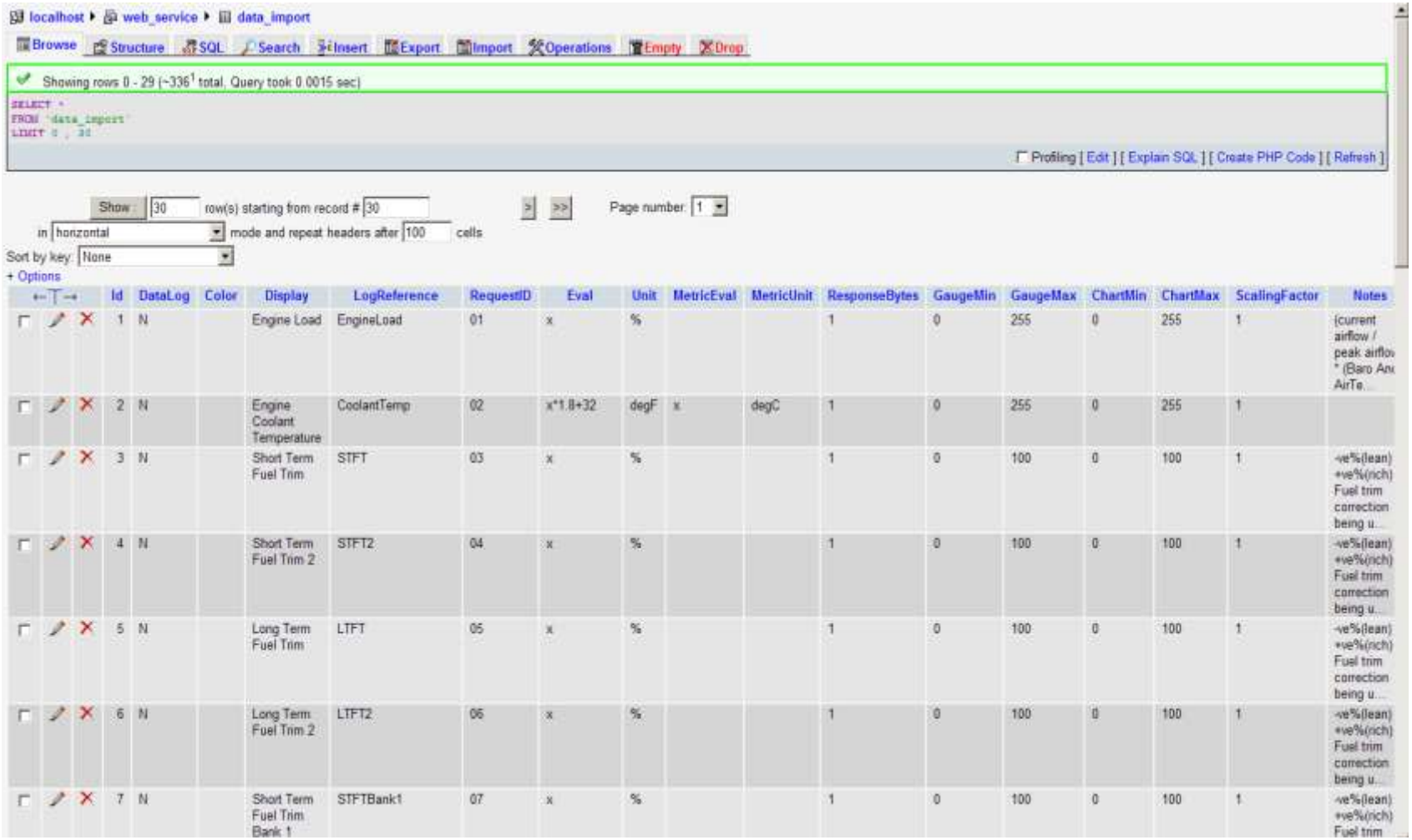
5.2.2 Smart Car Services Implementation in the Cloud

We implemented the core smart car services by building a SOAP server using PHP. The following tools are used to implement those web services:

- Web server : Internet Information Services 7 (IIS 7)
- PHP : PHP5
- Database : MySQL

We choose the Internet Information Services 7 (IIS 7) to provide the web services as the smart car web services deployment platform in Amazon EC2. The IIS7 is one of the Microsoft software products, and it has Windows Process Activation Service (WAS), which enables sites to use protocols other than HTTP and HTTPS. Besides, PHP applications can run on the Windows operating system in IIS7. The capabilities built into Windows Server 2008, such as FastCGI. FastCGI as an open protocol is supported by many open-source application frameworks on IIS 7. Before we publish our Web services on IIS7, we added a new web site in IIS 7. The SCIS, the CAN message service, follow, and the testing smart car services are deployed to the web site on IIS 7.A small script is written to connect CANoe with the services in the cloud so that the simulated CAN messages can be uploaded to the cloud.

As described in Chapter 4, the SCIS and the CAN message service will manage their respective databases in order to support the smart car services. For example, the CAN message service will manage the database that stores the CAN data. We choose MySQL as our database in Amazon EC2. Figure 5.8 below shows the CAN message file that is stored in the cloud. The CAN message file (generated by CANoe) is collected by SCIS and forwarded to the CAN message service for storage. Other smart car services can treat the CAN message service as a smart car information resource and invoke it to retrieve information when needed. Once the CAN message file is provided by smart car via SCIS, the CAN message service will check the status of the smart car via parameters diagnosis in the CAN message.



Id	DataLog	Color	Display	LogReference	RequestID	Eval	Unit	MetricEval	MetricUnit	ResponseBytes	GaugeMin	GaugeMax	ChartMin	ChartMax	ScalingFactor	Notes
1	N		Engine Load	EngineLoad	01	x	%			1	0	255	0	255	1	(current airflow / peak airflo * (Baro An AirTe
2	N		Engine Coolant Temperature	CoolantTemp	02	x*1.8+32	degF	x	degC	1	0	255	0	255	1	
3	N		Short Term Fuel Trim	STFT	03	x	%			1	0	100	0	100	1	-ve%(lean) +ve%(rich) Fuel trim correction being u
4	N		Short Term Fuel Trim 2	STFT2	04	x	%			1	0	100	0	100	1	-ve%(lean) +ve%(rich) Fuel trim correction being u
5	N		Long Term Fuel Trim	LTFT	05	x	%			1	0	100	0	100	1	-ve%(lean) +ve%(rich) Fuel trim correction being u
6	N		Long Term Fuel Trim 2	LTFT2	06	x	%			1	0	100	0	100	1	-ve%(lean) +ve%(rich) Fuel trim correction being u
7	N		Short Term Fuel Trim Bank 1	STFTBank1	07	x	%			1	0	100	0	100	1	-ve%(lean) +ve%(rich) Fuel trim

Figure 5.8 An Example of CAN message file was stored in the Cloud

In our designed platform, we choose the google service as our location service. It is a free service and is designed to be faster and more applicable to mobile devices [72]. We invoke the google map service in our designed platform as one of core smart car services, the google map service allows us to create robust map applications in the designed smart car services [80, 81]. For example, we can add and display detailed information about the car repair stations on the map, including addresses, icons, and ratings information. The google map service requires API key to identify our smart car service. Using the valid API key allows us to exceed anonymous limits by connecting requests back to our project. Our API key is shown as below:

API key: AIzaSyANoyPwwY6jcD9o8sgf7DTpdRV7N4ZGzp8

We use the goole map service to display the smart car's location as well. It relies on the identification of the geographic location of the smart car via the GPS sensor. We pass a required sensor parameter to the <script> tag to indicat that the smart car service is using the GPS sensor. The script we used for achieving this function is showen below:

```
<script type="text/javascript"
src="http://maps.googleapis.com/maps/api/js?sensor=true"></script>
```

This approach is an easy way to indicate that our smart car service is using a GPS receiver(sensor) to determine the user's location. In addition, we need the google map service to return information about places, such as locations of car repair staions around the smart car. To archieve this, we use HTTP requests to get the locations as latitude and longitude coordinates. There are three (3) kinds of place requests we use in our map service. Firstly, the place report request lets us add or delete places in the map. Secondly, by using the place search request, the google map service will return a list of nearby places based on the location

information from the smart car. Thirdly, the place details request can return more detailed information about a chosen place. We will show an example request test below, including a search for places of type, eg 'car repair station' within a 5000m radius of a point in Sydney, Australia, containing the word 'Kmart' in their name:

https://maps.googleapis.com/maps/api/place/search/xml?location=-33.8670522,151.1957362&radius=5000&types=car repair station&name=kmart&sensor=true&key=AIzaSyANoyPwwY6jcD9o8sgf7DTpdRV7N4ZGzp8

The parameter “xml” in the above example indicates that the output message as is in the XML format. Besides, we use some certain parameters for sending this place search request. The list of parameters and their possible values[72] are enumerated below.

- location (required) — The latitude/longitude around the smart car which to retrieve place information.
- radius (required) — The distance (meters) within.
- types (optional)
- language (optional)
- name (optional)
- sensor (required)
- key (required)

Once we choose an repair station from the place search response, we can request more details by sending another place details request. It will return more information about the chosen car repair station, such as address, phone number, user rating, etc. An example of a place details request is shown below. In this example, the “reference” is a required parameter, which is a textual identifier that uniquely identifies the chosen car repair station, as a response from the place search request.

https://maps.googleapis.com/maps/api/place/details/json?reference=CmRYAAAA
ciqGsTRX1mXRvuXSH2ErwW-
jCINE1aLiwP64MCWDN5vkXvXoQGPKldMfmdGyqWSpm7BEYCgDm-
iv7Kc2PF7QA7brMAwBbAcqMr5i1f4PwTpaovIZjysCEZTry8Ez30wpEhCNCX
pynextCld2EBsDkRKsGhSLayuRyFsex6JA6NPh9dyupoTH3g&sensor=true&key
= AIzaSyANoyPwwY6jcD9o8sgf7DTpdRV7N4ZGzp8

5.3 Summary

This chapter presented the implementation and simulation to apply the Cloud-based Information Integration Platform. The CANoe was used for simulate the ECU's in CAN bus network. We can trace and test applications by monitoring the CAN message traffic. Besides, we use Amazon EC2 to run our designed smart car services. In Amazon EC2 cloud, we use the IIS 7 to provide the web services deployment platform. In addition, the Google map service is chosen to act as the location service.

By using the smart car services in the Cloud-based Information Integration Platform, smart car applications will lend themselves very nicely to the users of smart cars. They are easy to invoke, produce a discretely formatted response, and support event-driven XML parsing which is less memory intensive than tree-based parsing. The smart car services interchange message formats are defined under the XML schema. To use the core smart car services, we connect the smart car infrastructure to collect CAN message, and the GPS information. By adopting this flexible SOA connections between core smart car services, we can take an existing business process and deliver it without much effort through a different business channel. In the next chapter, we will present a car repair service to demonstrate the designed platform.

Chapter 6

6 CASE STUDY

To illustrate the applicability of the proposed Smart Car Information Integration Platform, we present a case study about a car repair facilitation in this Chapter.

6.1 Car Repair Facilitation Scenario

The car repair facilitation scenario demonstrates the working process in the Smart Car Information Integration Platform. The car repair facilitation scenario starts receiving the input and request which from the smart car information service. Generally, the input includes CAN message file and location information. Once the CAN message service is invoked by SCIS, the SCIS will forward the CAN message file to the CAN message service. In the cloud, the CAN message file will be stored in the the cloud to analyse and judge the status of the smart car if it is normal. If not, the smart car repair service will find out where the mistake is, and if the smart car can continue moving. A Smart car user can request the roadside assistance service when they cannot drive the car. In this situation, the roadside

assistance company, as the service provider, will go to the user's place. The roadside assistance company can arrange their task based on received GPS location information and error message from the smart car. Sometimes the smart car users can continue to drive their cars to the nearby car repair station to seek help.

If the car can move, the car repair service will find out the correct car repair station by invoking the map service and car repair station service, and then the map service will provide the navigation to the user. The business process of this scenario is described as in Figure 6.1 below.

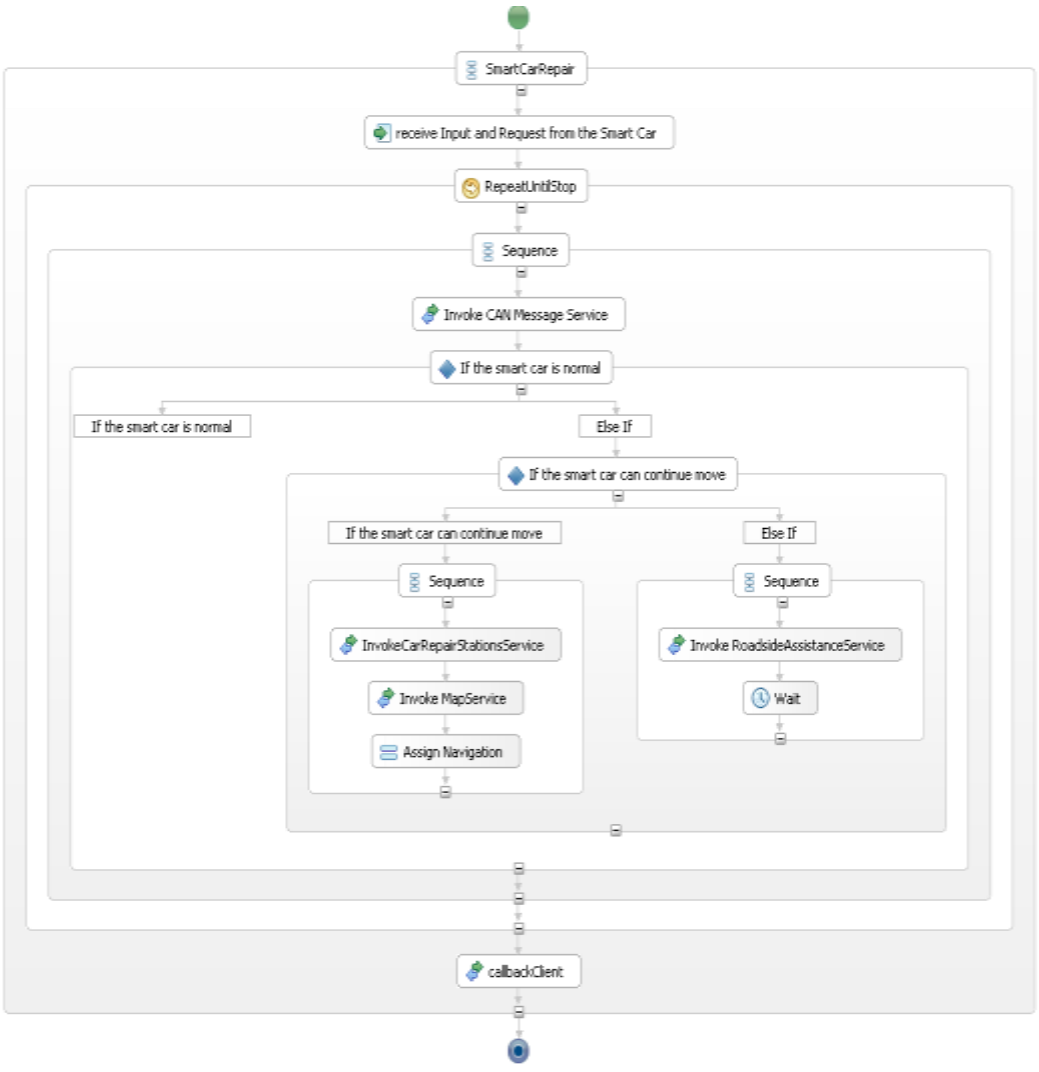


Figure 6.1 The Business Process of the Car Repair Facilitation Scenario

As we outlined above, in our designed platform the Web service interactions can be described in business processes. We now use Web Services Business Process Execution Language (WS-BPEL) which is a standard executable language for specifying actions within business processes with designed web services. Appendix A shows the logic part in BPEL which defines our car repair business process by using the XML-based language.

Our proposed car repair process in the BPEL exports and imports information by using smart car web service interfaces exclusively. Appendix B shows the imported client WSDL in our BPEL. It is clear to see that, the BPEL programming language provides facilities to enable sending and receiving messages from those smart car web services. Those partner links in the BPEL process is shown in appendix B.

The sample scenario is presented in this section to demonstrate the functions of the J1939 TestFeatureSet in CANoe. In this case scenario, an ECU will be tested for tyre pressure control. The process of this scenario is described as follows.

1. The sensor that monitors the tyre pressure will submit tyre pressure data to CAN bus. Once an abnormal tyre pressure is identified by CAN message service, the car repair service will be invoked to judge the status of the car to see if it is still safe to drive. At the same time, a warning message will be sent to the smart car information service to notify the user of the abnormality. After that, the car repair service can find all the details of the abnormal tyre, such as the the maker, the model and so on, based on collected error messages from the CAN message service. It is a completely automatic process. Meanwhile, the screen in the smart car displays the warning message to the driver. If the car is still safe to drive, the car repair service will prompt the driver to go to the car repair station.

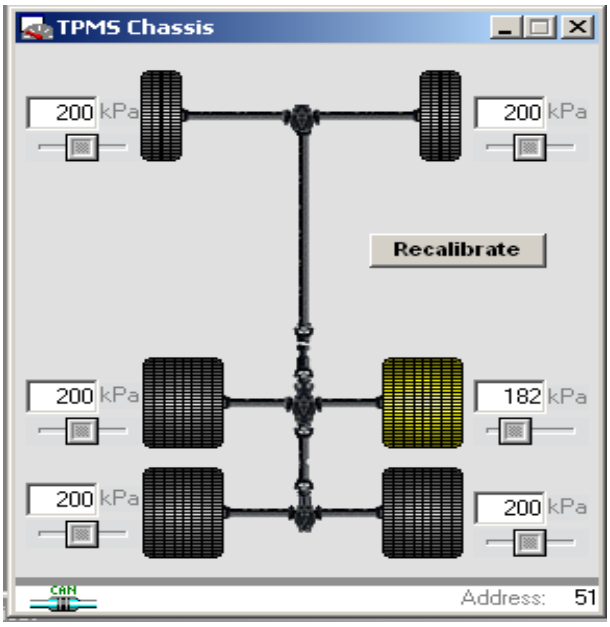


Figure 6.2 CANoe Simulation

2. We will use the stored repair stations information in the station database service as static context before the next step. The static context in the database includes station locations, parts categories, station names and reviews of products and services. The car repair service will call the Geocoder service [82] in this stage to prepare those car repair stations data for invoking. The station database service will be invoked to provide a list of available repair stations. By using the Google map service, we can overlay car the repair stations data in the map, such as creating several bookmarks of repair stations. The smart car users could choose one of car repair stations based on the price and experience. They just need to click a bookmark to open a balloon that contains the name, price and description of the chosen station, where the description is any valid HTML format information we want to display in the balloon. The navigation interface is shown in figure 6.3

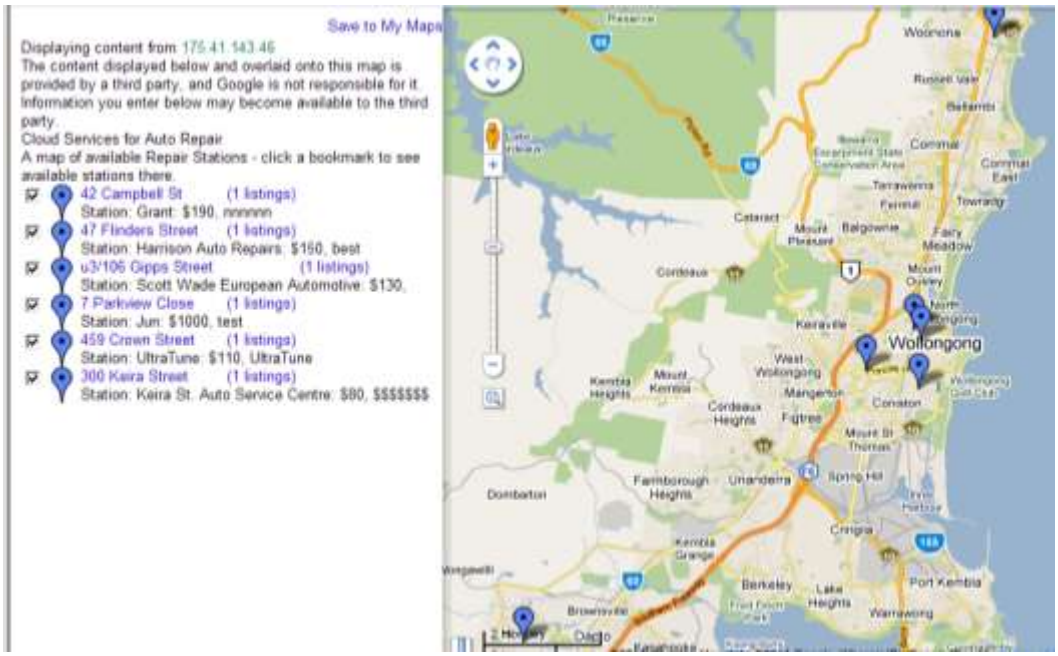


Figure 6.3 Navigation Service Map

3. The in-car computer collects the dynamic information from CAN bus and GPS receiver, then submits them to the smart car information service via wireless network. The dynamic context comes from the CAN bus messages, provides some information about travel range, time, broken car part and car location to user. Some other dynamic context likely time, day, data and weather condition will be collected from other sources.

The smart car information service will give you a handy navigation at the fine step in the scenario. The driver can view the route in the standard road map view or in satellite view, and it automatically switches to street view when the smart car gets close to the destination. After the user reaches the destination, the repair service process goes to the ordinary business soon.

6.2 Message Flow

To support the car repair facilitation scenario, a number of messages needs to be exchanged between the in car computer and services in the cloud in order to share information and assist decision making. An open and simple protocol is designed to govern the message exchange. The message flow in our scenario is shown in Figure 6.4.

Firstly, the SCIS will collect messages from the CAN bus in real time. Meantime, the message from the GPS device is provided to the smart car information service. The CAN message file and the location information will be collected to smart car information service periodically. The smart car information service will forward the CAN message file to the CAN message service for storage and diagnosis. If an abnormality is found as the result of the car diagnosis, the CAN message service will invoke the car repair service to process the error message. After that, the car repair service will notify the user of the abnormality and request the map service and station database service automatically. During this period, the user could choose their prefer car repair station based on the provided information from the station database service. The GPS location information will be provided to the car repair service to find nearby car repair stations.

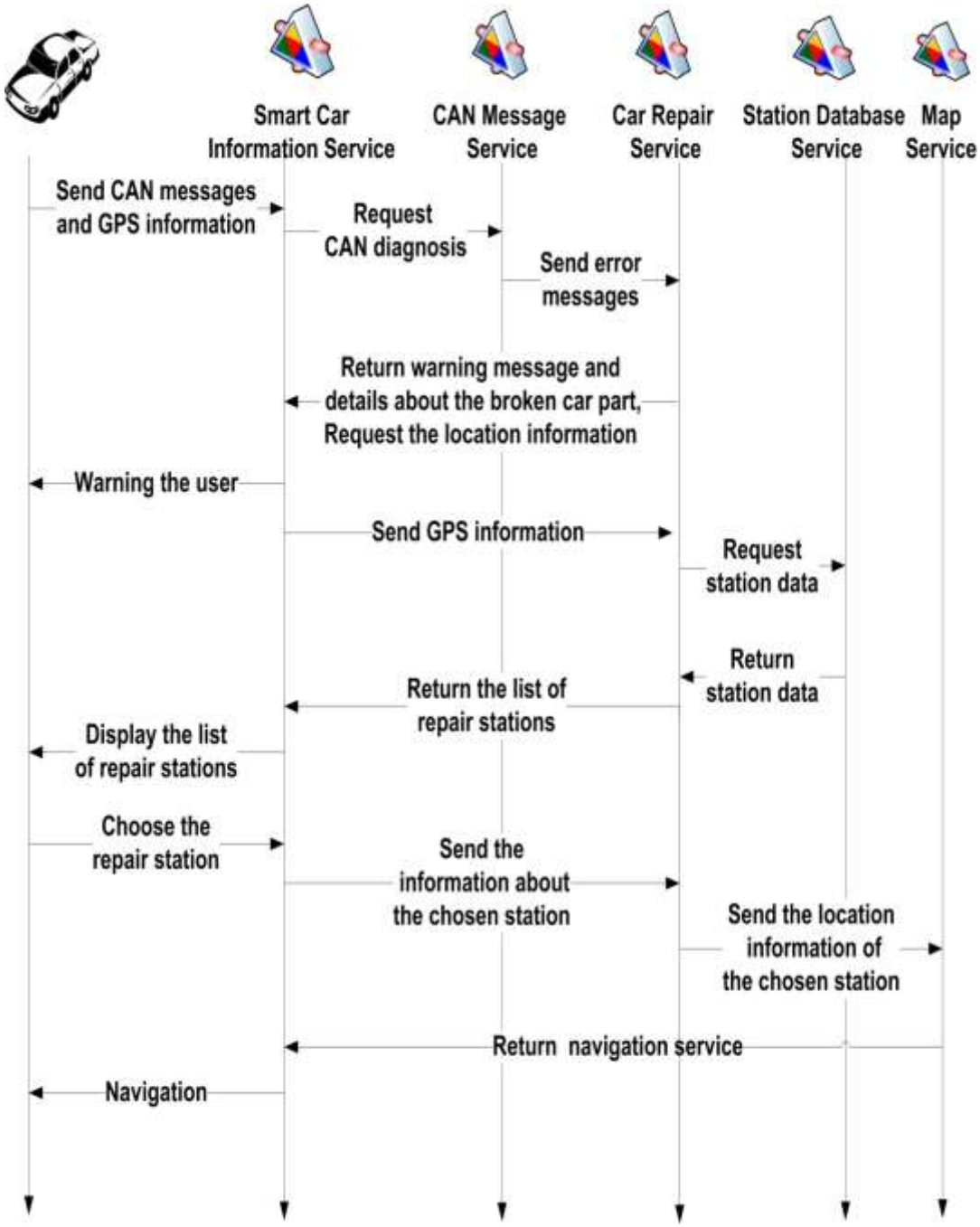


Figure 6.4 Message Flow of the Case Study

The objective of the car repair service is to establish if the car is still safe to drive and to find all the information to make the car be ready to be repaired. To achieve it, the car repair service will find out the car parts that are in need of repair via the CAN message service. With the car parts information, the car repair service will find an appropriate car repair stations list by querying a station database service. The selection of stations could be based on the location, user pre-specified preferences and feasibility of the travel range. Once the car repair station is chosen by the smart car user, the smart car information service will send the location information request to the map service. Finally, the location information is returned to the smart car information service and then to the smart car for information display and navigation.

6.3 Context in the Scenario

The tyre which is of low pressure is part of user's dynamic context. Other dynamic context elements include the maximum distance the car could travel, as well as the time and date used for checking the car repair stations' opening hours. Sometimes, drivers prefer to choose some stations based on their personal interest/s. These kinds of static preferences can help determine the list of nearby stations that will be displayed on in car computer screen.

Figure 6.5 shows an activity diagram for the scenario and the associated static and dynamic elements of the context.

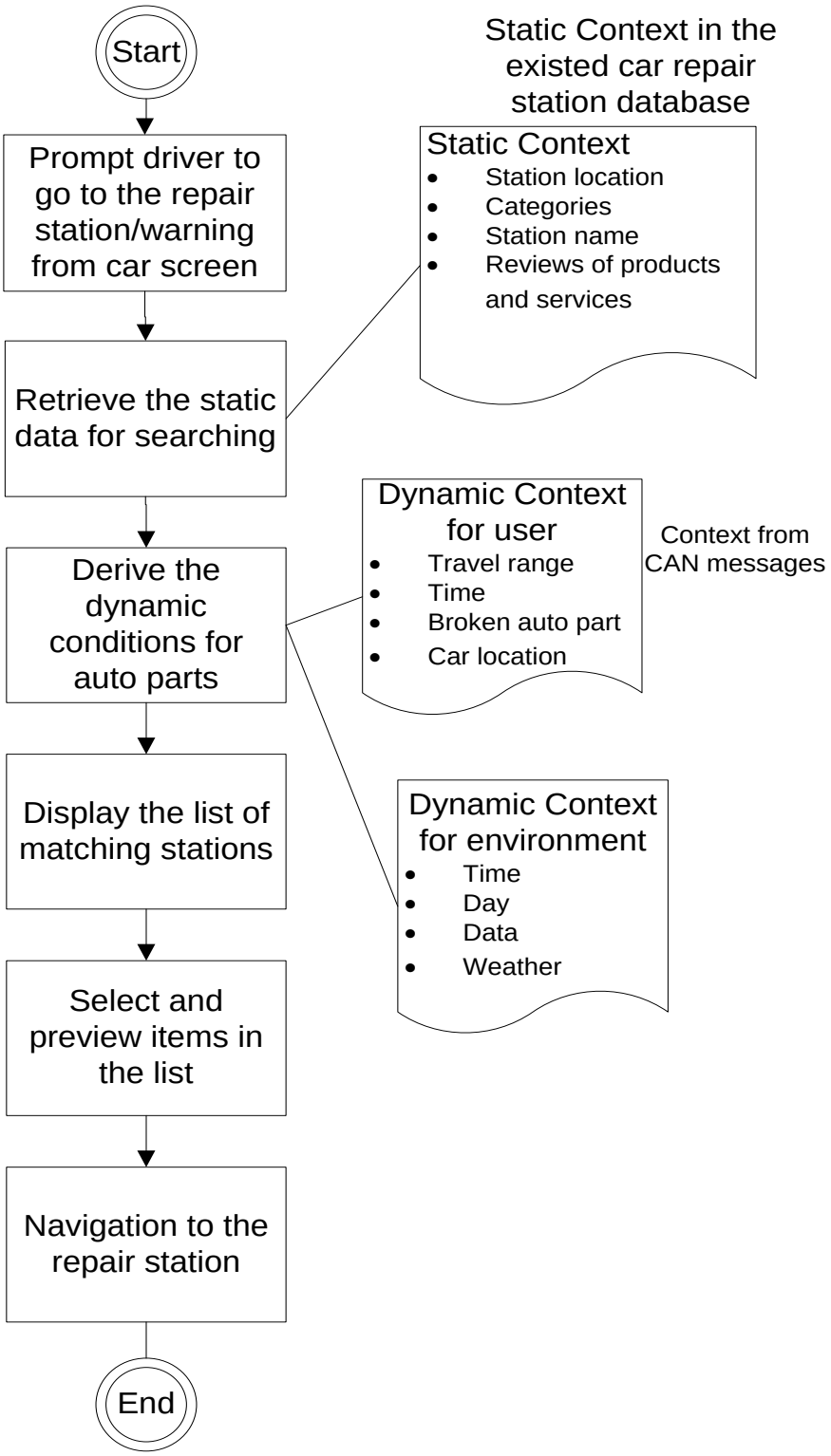


Figure 6.5Application Activity Diagram with Example Context Elements

The user-context contains the dynamic context and static context for smart car user. As dynamic context for user, car location, travel range, time and broken atuo parts are included. The details about the car location are street, city, suburb and zip. The travel range depends on the petrol level in the smart car. Normally, the sub-class time includes data, time of day and day of week. The information about the broken part is very important. In our designed scenario, we use name, type model and brand as four (4) fields in the subclass.

The RDFS graph in figure 6.6 depicts RDFS resources of user dynamic context in the described scenario.

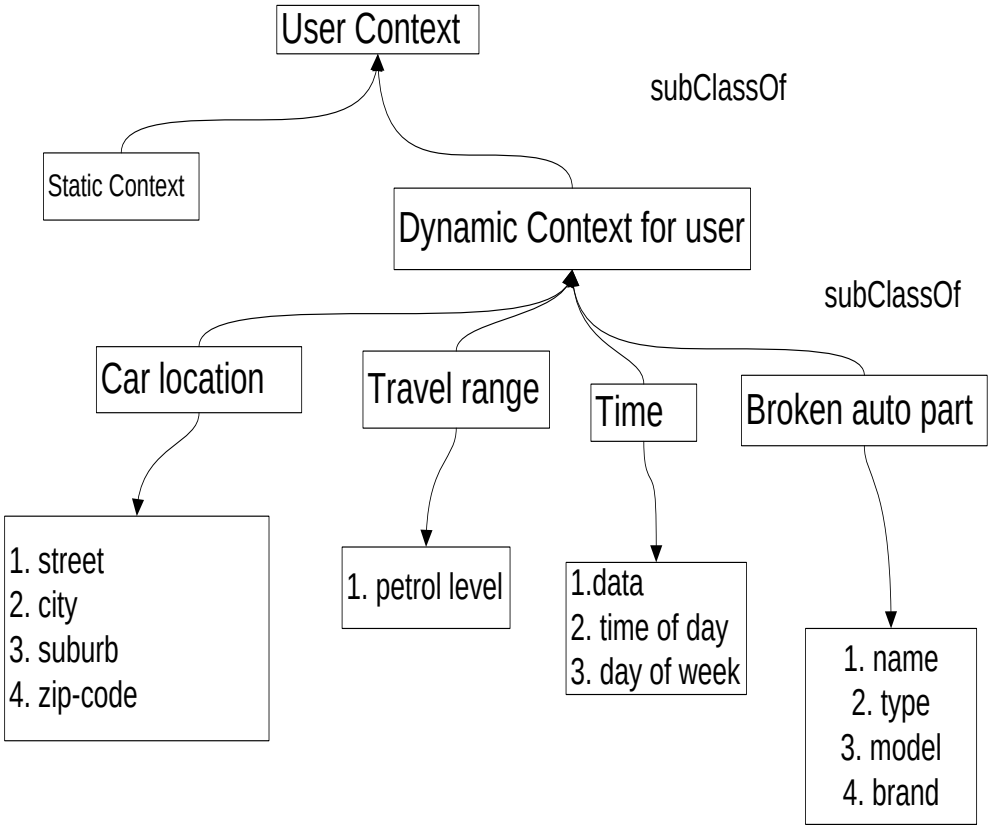


Figure 6.6 RDFS Graph of User Dynamic Context in Scenario

We store the static context in the car repair service. The RDFS graph in Figure 6.7 depicts RDFS resources of the user static context in the described scenario. The static context is another subclass of the user context. It has four (4) subclasses:

- station location,
- categories,
- station name and
- reviews.

There are four (4) fields in the station location information:

- location,
- city,
- suburb and
- zip code.

We put “category” and “e_coupons” as fields in the categories. The reviews of products and services include reviews, service guides and languages. All of the related data will be collected by receiving the request.

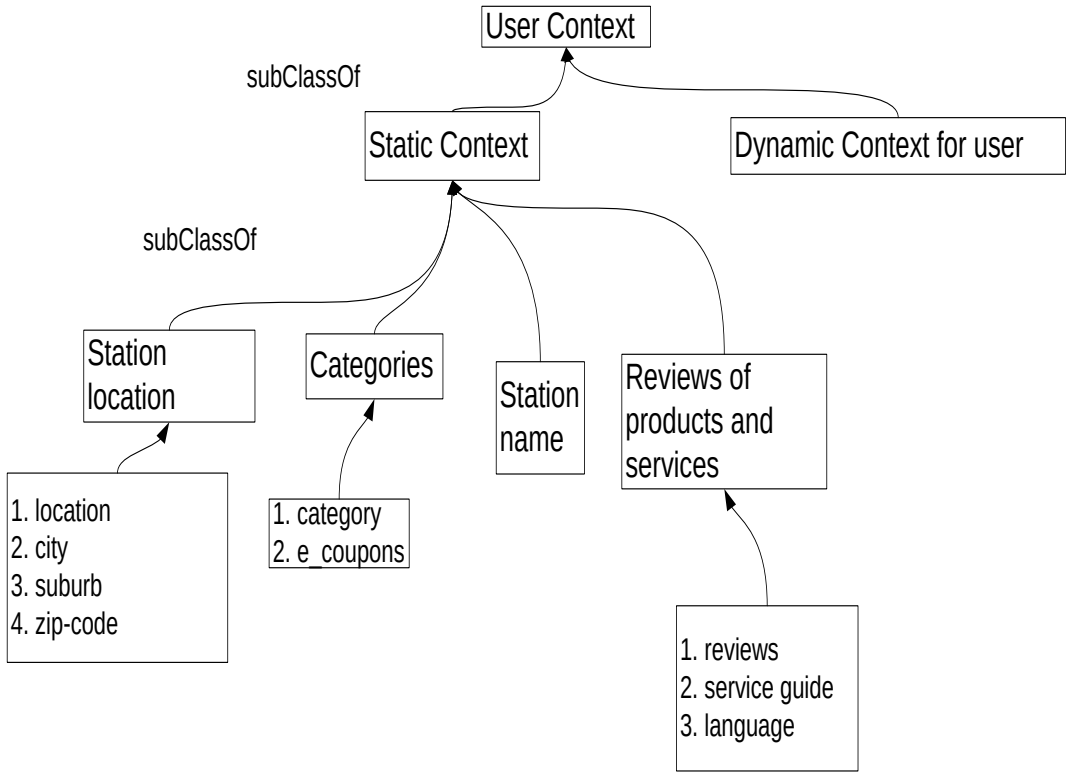


Figure 6.7 RDFS Graph of User Static Context in Scenario

The associated RDFS code of the user-dynamic context classes and properties in our case study is shown in appendix C. The associated RDFS code of the user static context classes and properties is shown in appendix D. We use RDF Schema to describe the ontologies in our car repair process. By adopting RDFS, a set of classes with certain properties using the RDF extensible knowledge representation language provides basic elements for the description of ontologies in the car repair process.

6.4 Web Service Interfaces Design

The car repair service is designed to support the scenario described in Section 6.1. Its implementation logic enables it to interact with other services in the cloud. As the user logins into the cloud platform via smart car information service, the car repair service needs to register with the smart car information service. The car repair service is one of the provided services in the cloud, and it is a virtualization service which will lets other smart car services be invoked with parameters and results.

```
- <definitions targetNamespace="http://175.41.143.46/soap/Car Repair Service">
  - <types>
    - <xsd:schema targetNamespace="http://175.41.143.46/soap/Car Repair Service">
        <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
        <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/"/>
      </xsd:schema>
    </types>
  - <message name="CarRepairSercieceRequest">
      <part name="request" type="xsd:string"/>
    </message>
  - <message name="CarRepairSercieceResponse">
      <part name="return" type="xsd:string"/>
    </message>
  - <portType name="Car Repair ServicePortType">
    - <operation name="CarRepairSercice">
        <documentation>Diagnoses</documentation>
        <input message="tns:CarRepairSercieceRequest"/>
        <output message="tns:CarRepairSercieceResponse"/>
      </operation>
    </portType>
  - <binding name="Car Repair ServiceBinding" type="tns:Car Repair ServicePortType">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    - <operation name="CarRepairSercice">
        <soap:operation soapAction="repair#repair" style="rpc"/>
      - <input>
          <soap:body use="encoded" namespace="repair" encodingStyle="http://schemas.xmlsoap.org
          /soap/encoding/"/>
        </input>
      - <output>
          <soap:body use="encoded" namespace="repair" encodingStyle="http://schemas.xmlsoap.org
          /soap/encoding/"/>
        </output>
      </operation>
    </binding>
  - <service name="Car Repair Service">
    - <port name="Car Repair ServicePort" binding="tns:Car Repair ServiceBinding">
        <soap:address location="http://175.41.143.46/webservices/carrepairservice.php"/>
      </port>
    </service>
</definitions>
```

List 6.1 Car Repair Service Interface

List 6.1 is the WSDL document that defines the interface of the smart car repair service. The WSDL document provides all the information about the car repair service in an XML document that can be read by a service client.

6.5 Location Data Transformation in Map Service

We use XSL and XSLT display transformations to deal with the location data in the map service. In our scenario, the Google Maps service is chosen to display the map and navigation.

Location is the most widely used context item in context-aware computing [8]. The location data could be used to display in an Earth browser such as Google Earth, Google Maps, and Google Maps for mobile. These applications use KML format file to store location data.

KML uses a tag-based structure with nested elements and attributes and it is based on the XML standard. All tags are case-sensitive and must be appear exactly as they are listed in the KML Reference [83]. The class tree for KML elements is shown as Figure 6.8 below. All elements derived from *Object* can have an **ID** assigned to them, and the **ID** is used by the KML update mechanism for files loaded with a NetworkLink [84]. It is also used by shared styles. The **ID** is a standard XML ID.

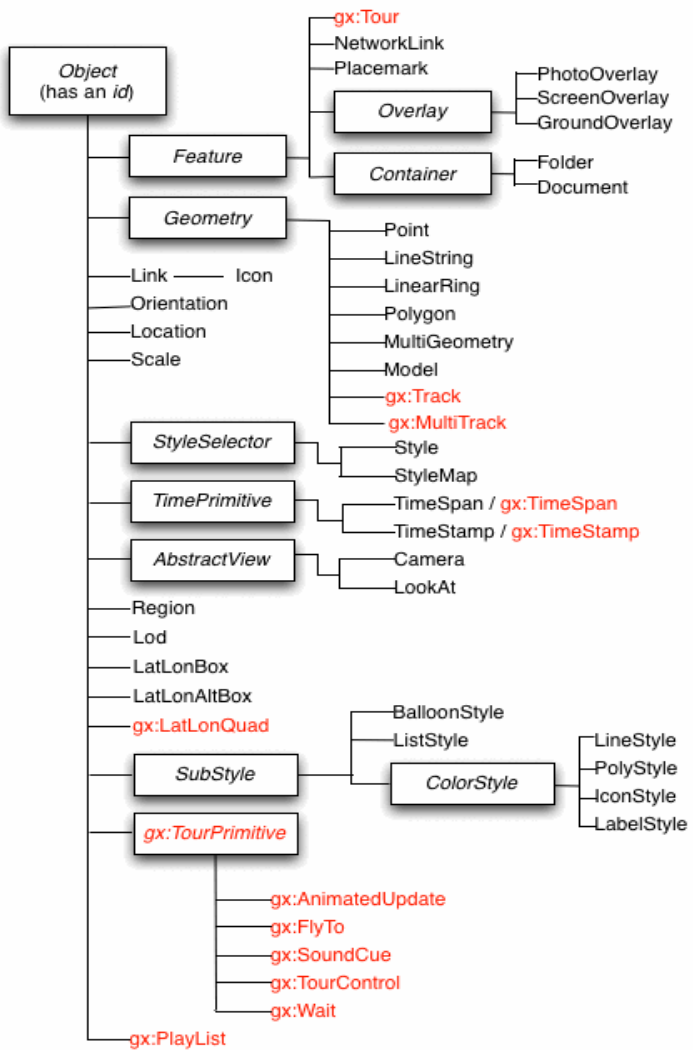


Figure 6.8 The Class Tree for KML Elements [84]

Originally, we get the location database information from a XML format in the cloud. The XML file is shown in List 6.10. We cannot use it to directly overlay the Google map service. With the database contents in XML format, we can use XSLT to transform the data into KML to produce data we can overlay on a map. After the transformation process, the XML file is changed to a standard KML. Thus, this KML file can be used to overlay the google map. The example of the XML file is shown in List 6.2.

```
- <listings>
  - <listing>
      <id>6</id>
      <address>42 Campbell St</address>
      <apt_no>Grant</apt_no>
      <city>Wollongong</city>
      <state>NSW</state>
      <zipcode />
      <longitude>150.9152110</longitude>
      <latitude>-34.3434540</latitude>
      <apt_type />
      <rent>190</rent>
      <notes>nnnnnn</notes>
    </listing>
  - <listing>
      <id>7</id>
      <address>47 Flinders Street</address>
      <apt_no>Harrison Auto Repairs</apt_no>
      <city>Wollongong</city>
      <state>NSW</state>
      <zipcode />
      <longitude>150.8929330</longitude>
      <latitude>-34.4187660</latitude>
      <apt_type />
      <rent>150</rent>
      <notes>best</notes>
    </listing>
  - <listing>
      <id>8</id>
      <address>u3/106 Gipps Street</address>
      <apt_no>Scott Wade European Automotive</apt_no>
      <city>Wollongong</city>
      <state>NSW</state>
      <zipcode />
      <longitude>150.8910490</longitude>
      <latitude>-34.4160200</latitude>
      <apt_type />
      <rent>130</rent>
      <notes />
    </listing>
  - <listing>
      <id>10</id>
      <address>459 Crown Street</address>
      <apt_no>UltraTune</apt_no>
      <city>Wollongong</city>
      <state>NSW</state>
      <zipcode />
      <longitude>150.8765812</longitude>
      <latitude>-34.4261033</latitude>
      <apt_type />
      <rent>110</rent>
      <notes>UltraTune</notes>
    </listing>
  - <listing>
      <id>11</id>
      <address>300 Keira Street</address>
      <apt_no>Keira St. Auto Service Centre</apt_no>
      <city>Wollongong</city>
      <state>NSW</state>
      <zipcode />
      <longitude>150.8922500</longitude>
      <latitude>-34.4305740</latitude>
      <apt_type />
      <rent>80</rent>
      <notes>$$$$$$$</notes>
    </listing>
  </listings>
```


List 6.2 Original Location Database Information from the Cloud in a XML File

```
<?xml version="1.0" ?>
- <kml xmlns="http://earth.google.com/kml/2.2">
  - <Document>
      <name>Cloud Services for Auto Repair</name>
      <description>A map of available Repair Stations - click a bookmark to see
        available stations there.</description>
    - <Placemark xmlns="">
        <name>42 Campbell St (1 listings)</name>
      - <description>
        - <![CDATA[
                    <div>
                  Station: Grant:
                  $190,
                  nnnnnn</div>

          ]]>
        </description>
      - <Point>
          <coordinates>150.9152110, -34.3434540</coordinates>
        </Point>
      </Placemark>
    - <Placemark xmlns="">
        <name>47 Flinders Street (1 listings)</name>
      - <description>
        - <![CDATA[
                    <div>
                  Station: Harrison Auto Repairs:
                  $150,
                  best</div>

          ]]>
        </description>
      - <Point>
          <coordinates>150.8929330, -34.4187660</coordinates>
        </Point>
      </Placemark>
    - <Placemark xmlns="">
        <name>u3/106 Gipps Street (1 listings)</name>
      + <description>
      - <Point>
          <coordinates>150.8910490, -34.4160200</coordinates>
        </Point>
      </Placemark>
    - <Placemark xmlns="">
        <name>459 Crown Street (1 listings)</name>
      - <description>
        - <![CDATA[
                    <div>
                  Station: UltraTune:
                  $110,
                  UltraTune</div>

          ]]>
        </description>
      - <Point>
          <coordinates>150.8765812, -34.4261033</coordinates>
        </Point>
      </Placemark>
    - <Placemark xmlns="">
        <name>300 Keira Street (1 listings)</name>
      - <description>
        - <![CDATA[
                    <div>
                  Station: Keira St. Auto Service Centre:
                  $80,
                  $$$$$$$</div>

          ]]>
        </description>
      - <Point>
          <coordinates>150.8922500, -34.4305740</coordinates>
        </Point>
      </Placemark>
  </Document>
</kml>
```

List 6.3 Location Information in a KML File

The List 6.3 above shows the location information in a KML file. This KML document contains name, description, and placemark elements. Each placemark will becomes a bookmark on a Google Map. Since Google Maps renders the description verbatim as HTML, each line must be wrapped in its own element. We include HTML as the description format.

An XSLT key creates an index by a set of elements matching single criteria. This operation will require that the style sheet have two (2) steps in the process:

- Collect a set of group leaders, one per unique car repair station address in the file, to create the placemark element.
- For each car repair station address, pass all listings with that address into a template to process. In our case, we render one element per station within the placemark.

The smart car information service finds addresses of repair stations based on the existing database. It then returns a point location that contains the latitude-longitude coordinates of those stations. Figure 6.9 is the result page of designed Smart Car Information Service. The SCIS can display the car type on the head of the result page. The smart car user could check the car status manually by viewing the details of the smart car. We can see that the map service is involved to display the locations of car repair stations. A list of nearby car repair stations is shown below the legends. By choosing the car repair station, the details of the chosen station can be seen in the map. The navigation service is prepared to direct the smart car user to the chosen car repair station.

Car Information Service

MITSUBISHI MAGNA TL 3.5 Lit. Sedan Auto 2003 View Details VIEW BY MAP

Name	Address	Price (AUD inc. gst)	Description
Wollongong Auto Excellence	257 Keira Street, WOLLONGONG DC	$172.29	4 5
David Carlon Motors	105-107 Princes Highway, UNANDERRA	$163.95	2 3
Practic Automotive Services	32 Albert St, CORRIMAL	$135.27	2 4
Gary's Mechanical Repairs & Service	Unit 2, 232 Shellharbour Rd, WARILLA	$162.91	2 2
Kelly's Car Repairs	1 Loftus Drive , BARRACK HEIGHTS	$172.95	1 2
Cougar Automotive	77 Old Lake Entrance Rd, OAK FLATS	$126.85	3 3
Avendel Automotive Service	3/113 Industrial Road , OAK FLATS	$158.45	2 2
ULTRA TUNE ALBION PARK	UNIT 4/29-31 DURGADIN DRIVE, ALBION PARK RAIL , ALBION PARK RAIL	$184.85	3 4
Helensburgh Car Services	187 Parkes Street, HELENSBURGH	$167.95	2 3
G.A.S	6 Maxwell Place, NARELLAN	$151.89	2 2

Figure 6.9 Search Result of Designed Service

6.6 Summary

This chapter described a car repair facilitation scenario to demonstrate the proposed Smart Car Information Integration Platform. Our case study use the SCIS as a smart car application access point to invoke the car repair service, CAN message service, station database service and map service to achieve the typical car repair facilitation for smart cars. This scenario proves the concept that the Cloud-based Information Integration Platform for Smart Cars could ease the intelligibility of smart cars with the core smart car services composition.

Chapter 7

7 CONCLUSIONS

7.1 Summary

This study presented a Cloud-based Information Integration Platform for Smart Cars. The main focus of this work is to leverage cloud computing and smart car technology to enable the information sharing and smart car service integration in the designed platform.

This thesis analysed the challenges and issues of the contemporary smart car technologies, and advocated that cloud computing, as the next generation of the computing paradigm, complements the smart car technology. Motivated by this finding, this research explored the architecture for Cloud-based Information Integration Platform. With this platform, two core services, the smart car information service and the CAN message service, were implemented to support to information integration. The Smart Car Information Service (SCIS) is acting as an interface at the heart of this platform. It provides the access point for the smart car user to access various smart car applications. The main functions provided by

the SCIS include: user profile management, smart car services registration and discovery, and smart car services access. The CAN message service is another core smart car services and was designed to process the CAN message file from smart cars periodically. Thus the CAN data can be managed as a smart car information resource in our designed platform for other smart car services.

An open and simple protocol was designed in the case study to share the CAN message data over the CAN message services. The CAN information which collected by CAN bus was shared with other smart car service in the cloud. We designed a CAN message conversion approach to apply the smart car information integration platform.

To demonstrate the proposed approach, we implemented the core services in Amazon EC2 and used CANoe to simulate the smart car environment. Based on the implementation, a case study was discussed to illustrate how a smart car repair service can be deployed within this framework to facilitate smart car faulty warning and repair. The implementation and case study confirm that we can use the current technologies to build the platform in smart cars. In the proposed platform, the cloud is used as the container for smart car services. Smart car applications are back-processed based on the current context of the smart car user. Communication can be minimised with the cloud. Our designed platform makes use of the CAN bus messages from smart cars to make smart car applications context aware. The information from the CAN bus can be uploaded into the cloud to support smart car services at all levels, thus achieving information sharing and integration.

7.2 Discussion

The proposed platform brings a number of potential benefits. The designed Cloud-based Information Integration Platform for Smart Cars is economical and practical for both car users and smart car service providers. By adopting SOA and SaaS in

Cloud-based Information Integration Platform, the cost of smart car applications, design, development, deployment and services support will be reduced. Meanwhile, smart car application developers can design different products by using common hardware standards to achieve flexibility, functionality, low cost and success. Cloud processing for smart cars allows smart car users to start small and increase budget only when there is an increase in their smart car services or applications needs. In addition, the Cloud-based Information Integration Platform for Smart Cars can help automotive manufacturers and their suppliers follow the rapid development of consumer-product technology.

Moreover, in our designed platform, smart car services can be enhanced with SOA to create new smart car services that exceed the capabilities of traditional software in the in-car computer. The standards provide automated service-oriented ways in which composite solutions come together with less work, and services ensure there is useful content from which to draw. Combining these initiatives it creates the opportunity to develop a completely new paradigm of smart car services in the cloud. The Cloud-based Information Integration Platform for Smart Cars provides a very flexible solution for building one's own custom objects and UI. This enables customers to model their business processes which are unique to them.

We recognise that there some limitations in our research. The major research limitation of this study is the failure to collect and analyse data that would yield the relationship between smart cars and an ICT-based system. Furthermore, we did not address the reliability and security issues in the Cloud-based Information Integration Platform. In the design of the platform and smart car core services, performance was not considered. In the real environment, some factors will impact on the performance of the designed platform, such as the network speed, which might not be fast enough in some areas.

Besides, we cannot make causal conclusions from the case study, because it is very hard to rule out alternative explanations without real road tests and research. Our case study involves the behavior of the business process of

car repair. Only one case study may not reflect the behavior of the whole information integration for smart cars.

7.3 Future Work

Future work is underway to further explore the services and software layer in the designed platform. Different cloud service deployment models will be considered in details as well. The methods used in this study appear to be rather case-specific and possible extensions and applications of these methods to generic design cases require further study.

REFERENCES

1. Davis, R., et al., *Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised.* Real-Time Systems, 2007. **35**(3): p. 239-272.
2. *Why would anyone want a computer in their vehicle?* 2011 [cited; Available from: http://www.mp3car.com/content/150-car-pc-101.html.
3. Rosen, M., et al., *Applied SOA : Service-Oriented Architecture and Design Strategies*. 2008, Hoboken, NJ, USA: John Wiley & Sons.
4. Jie, J., X. Hai-feng, and Z. Ling. *The Design for General-purpose Interface of Community Informatization Basic Database Based on SOA*. in *E-Business and E-Government (ICEE), 2010 International Conference on*. 2010.
5. Sarjoughian, H., et al., *A simulation framework for service-oriented computing systems*, in *Proceedings of the 40th Conference on Winter Simulation*. 2008, Winter Simulation Conference: Miami, Florida.
6. O'Brien, L., P. Brebner, and J. Gray, *Business transformation to SOA: aspects of the migration and performance and QoS issues*, in *Proceedings of the 2nd international workshop on Systems development in SOA environments*. 2008, ACM: Leipzig, Germany.
7. Bean, J., *SOA and Web Services Interface Design : Principles, Techniques, and Standards*. 2009, Burlington, MA, USA: Morgan Kaufmann.
8. Pashtan, A., *Mobile Web Services*. 2005, Cambridge: Cambirdge University Press.
9. Keyvan, M., et al., *A comparative evaluation of semantic web service discovery approaches*, in *Proceedings of the 12th International Conference on Information Integration and Web-based Applications \&\#38; Services*. 2010, ACM: Paris, France.
10. Mojtaba, K., et al., *An evaluation of state-of-the-art approaches for web service selection*, in *Proceedings of the 12th International Conference on Information Integration and Web-based Applications and Services*. 2010, ACM: Paris, France.
11. Adams, C. and S. Boeyen, *UDDI and WSDL extensions for Web service: a security framework*, in *Proceedings of the 2002 ACM workshop on XML security*. 2002, ACM: Fairfax, VA.
12. Luca, C., et al., *Synthesizing adapters for conversational web-services from their WSDL interface*, in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. 2010, ACM: Cape Town, South Africa.
13. Zimmermann, O., et al., *Service-oriented architecture and business process choreography in an order management scenario: rationale,*

concepts, lessons learned, in *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. 2005, ACM: San Diego, CA, USA.

14. Christensen, E., et al. *Web Services Description Language (WSDL) 1.1*. 2001 [cited; Available from: http://www.w3.org/TR/wsdl.
15. Phan, K.A., Z. Tari, and P. Bertok, *A benchmark on soap's transport protocols performance for mobile applications*, in *Proceedings of the 2006 ACM symposium on Applied computing*. 2006, ACM: Dijon, France.
16. Linthicum, D.S., *Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide*. 2009: Addison-Wesley Professional.
17. Youry, K. and V. Volodymyr, *Cloud computing infrastructure prototype for university education and research*, in *Proceedings of the 15th Western Canadian Conference on Computing Education*. 2010, ACM: Kelowna, British Columbia, Canada.
18. *Above the Clouds: A Berkeley View of Cloud Computing*, in *Technical Report*, M. Armbrust, et al., Editors. 2009: Berkeley.
19. Gonzalo, H.-C. and L. Dongman, *A virtual cloud computing provider for mobile devices*, in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing and Services: Social Networks and Beyond*. 2010, ACM: San Francisco, California.
20. Scurto, A., *Cloud Computing Extends SOA Capability*. National Underwriter / Property & Casualty Risk & Benefits Management, 2008. **112**(9): p. 18-19.
21. *Amazon Elastic Compute Cloud (Amazon EC2)*. 2010 [cited; Available from: http://aws.amazon.com/ec2/.
22. Shufen, Z., et al. *Analysis and Research of Cloud Computing System Instance*. in *Future Networks, 2010. ICFN '10. Second International Conference on*. 2010.
23. Buyya, R., et al. *Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities*. in *10th IEEE International Conference on High Performance Computing and Communications*. 2008. Dalian, PEOPLES R CHINA: Ieee Computer Soc.
24. *Amazon Elastic Compute Cloud*. 2010 [cited; Available from: http://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud.
25. Gabriela, T., F. Ian, and N. Svetlozar, *Reshaping text data for efficient processing on Amazon EC2*, in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. 2010, ACM: Chicago, Illinois.
26. Qunying, H., et al., *Cloud computing for geosciences: deployment of GEOSS clearinghouse on Amazon's EC2*, in *Proceedings of the ACM SIGSPATIAL International Workshop on High Performance and Distributed Geographic Information Systems*. 2010, ACM: San Jose, California.
27. Gideon, J., et al., *Data Sharing Options for Scientific Workflows on Amazon EC2*, in *Proceedings of the 2010 ACM/IEEE International*

Conference for High Performance Computing, Networking, Storage and Analysis. 2010, IEEE Computer Society.
28. Anand, V.H., S. Balasubrahmanya, and V.H. Raghu, *Level-4 SaaS applications for healthcare industry*, in *Proceedings of the 2nd Bangalore Annual Compute Conference*. 2009, ACM: Bangalore, India.
29. *Software as a service*. 2011.
30. Nitu, *Configurability in SaaS (software as a service) applications*, in *Proceedings of the 2nd India software engineering conference*. 2009, ACM: Pune, India.
31. *What is SAAS?* 2010 [cited; Available from: http://www.whatissaas.net/.
32. *Car Functions* IEEE Spectrum.
33. Milliron, T. and F. Behmaram-Mosavat, *Smart cars: driving the characters in Cars*, in *ACM SIGGRAPH 2006 Sketches*. 2006, ACM: Boston, Massachusetts.
34. Raymond, C. and C. Vinny, *System support for smart cars: requirements and research directions*, in *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*. 2000, ACM: Kolding, Denmark.
35. Oxlade, C., *Cars Inside and Out* 2009, New York: The Rosen Publishing Group.
36. Hu, J. and G. Li. *CAN-based passenger car starter information integrated control method and its implementation*. in *Industrial Electronics, 2009. ISIE 2009. IEEE International Symposium on*. 2009.
37. *i2010: Intelligent Car*. 2007 [cited; Available from: http://ec.europa.eu/information_society/activities/intelligentcar/technologies/index_en.htm.
38. Hemadri, B.V. and P.U. Kulkarni, *Detection and recognition of mandatory and cautionary road signals using unique identifiable features*, in *Proceedings of the International Conference \&\#38; Workshop on Emerging Trends in Technology*. 2011, ACM: Mumbai, Maharashtra, India.
39. Herkersdorf, A. and W. Stechele, *AutoVision: flexible processor architecture for video-assisted driving*, in *Proceedings of the conference on Design, automation and test in Europe: Proceedings*. 2006, European Design and Automation Association: Munich, Germany.
40. McCall, J. and M.M. Trivedi, *Driver monitoring for a human-centered driver assistance system*, in *Proceedings of the 1st ACM international workshop on Human-centered multimedia*. 2006, ACM: Santa Barbara, California, USA.
41. Minoura, K., N. Sekiyama, and T. Watanabe, *Estimation of blind spot based on driver's view*, in *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*. 2011, ACM: Seoul, Korea.
42. Wu, C.-Y., et al., *Automatic vehicle parking system using gyroscope*, in *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*. 2009, ACM: Suwon, Korea.

43. *Intelligent car*. 2011 [cited; Available from: http://en.wikipedia.org/wiki/Intelligent_car.
44. Bako, R., et al., *Trust, cognitive control, and control: the case of drivers using an Auto-Adaptive Cruise Control*, in *Proceedings of the 13th Eurpoean conference on Cognitive ergonomics: trust and control in complex socio-technical systems*. 2006, ACM: Zurich, Switzerland.
45. Kyo, S. and S.i. Okazaki, *In-vehicle vision processors for driver assistance systems*, in *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*. 2008, IEEE Computer Society Press: Seoul, Korea.
46. Ivanov, R., *Automatic GPS-based vehicle tracking and localization information system*, in *Proceedings of the 4th international conference conference on Computer systems and technologies: e-Learning*. 2003, ACM: Rousse, Bulgaria.
47. Biswas, S., et al., *Memory overflow protection for embedded systems using run-time checks, reuse and compression*, in *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*. 2004, ACM: Washington DC, USA.
48. Matthias, M., et al., *Design of an automotive traffic sign recognition system targeting a multi-core SoC implementation*, in *Proceedings of the Conference on Design, Automation and Test in Europe*. 2010, European Design and Automation Association: Dresden, Germany.
49. Yoon, J., B. Noble, and M. Liu, *Surface street traffic estimation*, in *Proceedings of the 5th international conference on Mobile systems, applications and services*. 2007, ACM: San Juan, Puerto Rico.
50. Harper, G.D.J., *Build Your Own Car PC*. 2006, Sydney: McGraw-Hill.
51. Leen, G. and D. Heffernan, *Expanding automotive electronic systems.* Computer, 2002. **35**(1): p. 88-93.
52. Sander, O., et al., *System concept for an FPGA based real-time capable automotive ECU simulation system*, in *Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes*. 2009, ACM: Natal, Brazil.
53. Lee, J.-C. and T.-M. Han, *ECU configuration framework based on AUTOSAR ECU configuration metamodel*, in *Proceedings of the 2009 International Conference on Hybrid Information Technology*. 2009, ACM: Daejeon, Korea.
54. Paret, D., *Multiplexed networks for embedded systems : CAN, LIN, flexray, safe-by-wire...* 2007, Chichester: John Wiley & Sons, Ltd.
55. Çenesİz, N. and M. Esin, *Controller area network (CAN) for computer integrated manufacturing systems.* Journal of Intelligent Manufacturing, 2004. **15**(4): p. 481-489.
56. Tindell, K. and A. Burns. *Introduction Guaranteeing Message Latencies on Control Area Network (CAN).* [cited; Available from: http://www.control.aau.dk/~jdn/edu/courses/e07/distrsys/can-iccb-tb-sep94.pdf.

57. Murphy, N., *A Short trip on the CAN bus.* Embedded Systems Programming, 2003. **16**(9): p. 9-+.
58. Osch, M.v. and S.A. Smolka, *Finite-state analysis of the CAN bus protocol*, in *Sixth Ieee International Symposium on High Assurance Systems Engineering*. 2001, Ieee Computer Soc: Los Alamitos. p. 42-52.
59. Popescu-Zeletin, R., *Implementing the ISO-OSI reference model*, in *Proceedings of the eighth symposium on Data communications*. 1983, ACM: North Falmouth, Massachusetts, United States.
60. Bret, H., et al., *CarTel: a distributed mobile sensor computing system*, in *Proceedings of the 4th international conference on Embedded networked sensor systems*. 2006, ACM: Boulder, Colorado, USA.
61. *Capabilities of our vehicle computers*. 2010 [cited; Available from: http://www.in-carpc.co.uk/features.htm.
62. Schmidt, A., et al., *Automotive user interfaces: human computer interaction in the car*, in *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*. 2010, ACM: Atlanta, Georgia, USA.
63. Hull, B., et al., *CarTel: a distributed mobile sensor computing system*, in *Proceedings of the 4th international conference on Embedded networked sensor systems*. 2006, ACM: Boulder, Colorado, USA.
64. NaviGadget. *In dash car PC with GPS Navigator with DVD and more*. 2006 [cited; Available from: http://www.navigadget.com/index.php/2006/06/30/in-dash-car-pc-with-gps-navigator-with-dvd-and-more.
65. Aigner, O. *CarPC Basics*. 2011 [cited; Available from: http://www.cartft.com/community/dockingstation/carpcbasics.
66. Keon, J., et al., *3G and 3.5G wireless network performance measured from moving cars and high-speed trains*, in *Proceedings of the 1st ACM workshop on Mobile internet through cellular networks*. 2009, ACM: Beijing, China.
67. Liao, Z., *Real-time taxi dispatching using Global Positioning Systems.* Commun. ACM, 2003. **46**(5): p. 81-83.
68. Peter, H.D., *Global Positioning System (GPS) Time Dissemination for Real-Time Applications.* Real-Time Syst., 1997. **12**(1): p. 9-40.
69. Gupta, C.D. *Application of GPS and Infrared for Car Navigation in Foggy Condition to Avoid Accident*. in *Computer Engineering and Applications (ICCEA), 2010 Second International Conference on*. 2010.
70. Shinder, D.L., *Smartphone location and navigation services.* 2010.
71. *Google Maps*. 2010 [cited; Available from: http://en.wikipedia.org/wiki/Google_Maps.
72. *Maps User Guide*. 2011 [cited; Available from: http://maps.google.com/support/bin/static.py?hl=en&page=guide.cs&guide=21670&from=21670&rd=1.
73. *Touchscreen*. 2011 [cited; Available from: http://en.wikipedia.org/wiki/Touchscreen.

74. *Voice command device*. 2011 [cited; Available from: http://en.wikipedia.org/wiki/Voice_control.
75. Pailer, R., F. Wegscheider, and S. Bessler, *A Terminal-Based Location Service Enabler for the IP Multimedia Subsystem*. 2006.
76. Google. *Google Maps Javascript API V3 Basics*. 2011 [cited; Available from: http://code.google.com/intl/en/apis/maps/documentation/javascript/basics.html.
77. *Vector*. 2011 [cited; Available from: http://www.vector.com/vi_index_en.html.
78. CANopen, *CANoe simulation and test tool*.
79. Vector. *Solutions for your CANopen Networking*. 2010 [cited; Available from: http://www.canopen-solutions.com/canopen_canoe_en.html.
80. Susan, D., *Create interactive web illustrations with google maps*, in *Proceedings of the 38th annual fall conference on SIGUCCS*. 2010, ACM: Norfolk, Virginia, USa.
81. James, D.T., *A Dijkstra's algorithm shortest path assignment using the Google Maps API: poster session.* J. Comput. Small Coll., 2010. **25**(6): p. 253-255.
82. Gilmore, J. *Introducing Google's Geocoding Service*. 2006 [cited; Available from: http://www.developer.com/lang/jscript/article.php/3615681.
83. *KML Tutorial*. 2011 [cited; Available from: http://code.google.com/apis/kml/documentation/kml_tut.html.
84. *KML Reference*. 2011 [cited; Available from: http://code.google.com/apis/kml/documentation/kmlreference.html.

LIST OF APPENDICES

Appendix A Orchestration Logic in BPEL

```
<bpel:sequence name="SmartCarRepair">

<!-- Receive input from requestor. Note: This maps to operation defined in
SmartCarInformation.wsdl  -->
 <bpel:receive name="receive Input and Request from the Smart Car Information Service"
partnerLink="client"   createInstance="yes" operation="initiate"
portType="tns:SmartCarInformation"/>

<!-- Asynchronous callback to the requester. Note: the callback location and correlation id is
transparently handled using WS-addressing.  -->

 <bpel:repeatUntil name="RepeatUntilStop"><bpel:sequence><bpel:invoke name="Invoke CAN
Message Service" partnerLink="client">

</bpel:invoke>
<bpel:if name="If the smart car is normal">
<bpel:throw name="Throw"></bpel:throw>
<bpel:elseif>
<bpel:if name="If the smart car can continue move">
<bpel:sequence name="Sequence">

<bpel:invoke name="InvokeCarRepairService"></bpel:invoke>
<bpel:invoke name="InvokeCarRepairStationsService"></bpel:invoke>
<bpel:invoke name="Invoke MapService"></bpel:invoke>
<bpel:assign validate="no" name="Assign Navigation"></bpel:assign>
</bpel:sequence>
<bpel:elseif>
<bpel:sequence>
<bpel:invoke name="Invoke RoadsideAssistanceService"></bpel:invoke>
<bpel:wait name="Wait"></bpel:wait>
</bpel:sequence>
</bpel:elseif></bpel:if>
</bpel:elseif>
</bpel:if>
<bpel:exit name="Exit"></bpel:exit>
</bpel:sequence>
</bpel:repeatUntil>
</bpel:sequence>
</bpel:process>
```

Appendix B List of Services Participating in the BPEL Process

```
<bpel:partnerLink name="client"
                partnerLinkType="tns:SmartCarInformation"
                myRole="SmartCarInformationProvider"
                partnerRole="SmartCarInformationRequester" />
<bpel:partnerLink name="CANmessageService" partnerLinkType="tns:CANmessageService"
          myRole="CANmessage"></bpel:partnerLink>
 <bpel:partnerLink name="CarRepairStationsService"
          partnerLinkType="tns:CarRepairStationsService" myRole="CarRepairStations"
          partnerRole="CarRepairStationsService" initializePartnerRole="no"></bpel:partnerLink>
<bpel:partnerLink name="MapService" partnerLinkType="tns:MapService"
          myRole="MapService"></bpel:partnerLink>
<bpel:partnerLink name="RoadsideAssistanceService"
          partnerLinkType="tns:RoadsideAssistanceService"
          myRole="RoadsideAssistanceService"
          partnerRole="RoadsideAssistance"></bpel:partnerLink>
<bpel:partnerLink name="CarRepairService" partnerLinkType="tns:CarRepair"
          myRole="CarRepair" partnerRole="CarRepair"></bpel:partnerLink>
   </bpel:partnerLinks>
```

Appendix C RDFS Code of User Dynamic Context in the Scenario


```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdf:Description rdf:about="ID:User-Context">
<rdf:type><rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-
schema#Class"/></rdf:type></rdf:Description>
<rdf:Description rdf:about="ID:User-Dynamic-Context">
<rdf:type><rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-
schema#Class"/></rdf:type><rdfs:subClassOf>
<rdf:Description rdf:about="ID:User-Context"/></rdfs:subClassOf>
</rdf:Description>
<rdf:Description rdf:about="ID:User-Static-Context">
<rdf:type><rdf:Description  rdf:about="http://www.w3.org/2000/01/rdf-schema#Class"/></rdf:type>
<rdfs:subClassOf><rdf:Description rdf:about="ID:User-Context"/>
</rdfs:subClassOf></rdf:Description>
<rdf:Description rdf:about="ID:Car-Location">
<rdf:type><rdf:Description  rdf:about="http://www.w3.org/2000/01/rdf-schema#Class"/></rdf:type>
<rdfs:subClassOf>
<rdf:Description rdf:about="ID:User-Dynamic-Context"/>
</rdfs:subClassOf>
</rdf:Description>
<rdf:Description rdf:about="ID:Time">
<rdf:type><rdf:Description  rdf:about="http://www.w3.org/2000/01/rdf-schema#Class"/></rdf:type>
<rdfs:subClassOf>
<rdf:Description rdf:about="ID:User-Dynamic-Context"/>
</rdfs:subClassOf>
</rdf:Description><rdf:Description rdf:about="ID:Broken-auto-part"><rdf:type><rdf:Description
rdf:about="http://www.w3.org/2000/01/rdf-schema#Class"/></rdf:type>
<rdfs:subClassOf>
<rdf:Description rdf:about="ID:User-Dynamic-Context"/>
```

</rdfs:subClassOf>
</rdf:Description>
<rdf:Description rdf:about="ID:address"><rdf:type><rdf:Description
rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/></rdf:type>
<rdfs:domain> <rdf:Description rdf:about="ID:Car-Location"/></rdfs:domain>
<rdfs:range><rdf:Description rdf:about="ID:Environment-Location">
<rdf:type><rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-
schema#Class"/></rdf:type></rdf:Description></rdfs:range></rdf:Description>
<rdf:Description rdf:about="ID:petrol_level"><rdf:type><rdf:Description
rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/></rdf:type><rdfs:domain>
<rdf:Description rdf:about="ID:Travel-range"/></rdfs:domain></rdf:Description>
<rdf:Description rdf:about="ID:time">
<rdf:type><rdf:Description rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/></rdf:type>
<rdfs:domain> <rdf:Description rdf:about="ID:Time"/></rdfs:domain>
<rdfs:range><rdf:Description rdf:about="ID:Enviroment-Time"><rdf:type>
<rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-
schema#Class"/></rdf:type></rdf:Description></rdfs:range></rdf:Description>
<rdf:Description rdf:about="ID:part"><rdf:type><rdf:Description
rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/></rdf:type>
<rdfs:domain>
<rdf:Description rdf:about="ID:Broken-auto-part"/></rdfs:domain></rdf:Description>
<rdf:Description rdf:about="ID:Travel-range">
<rdf:type><rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/></rdf:type>
<rdfs:subClassOf><rdf:Description rdf:about="ID:User-Dynamic-Context"/></rdfs:subClassOf>
</rdf:Description>
</rdf:RDF>

Addendix D RDFS Code of User Static Context in the Scenario

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdf:Description rdf:about="ID:User-Context">
<rdf:type><rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-
        schema#Class"/></rdf:type></rdf:Description>
<rdf:Description rdf:about="ID:User-Dynamic-Context">
<rdf:type><rdf:Description  rdf:about="http://www.w3.org/2000/01/rdf-schema#Class"/></rdf:type>
<rdfs:subClassOf><rdf:Description rdf:about="ID:User-
        Context"/></rdfs:subClassOf></rdf:Description>
<rdf:Description rdf:about="ID:User-Static-Context">
<rdf:type><rdf:Description  rdf:about="http://www.w3.org/2000/01/rdf-schema#Class"/></rdf:type>
<rdfs:subClassOf><rdf:Description rdf:about="ID:User-
        Context"/></rdfs:subClassOf></rdf:Description>
<rdf:Description rdf:about="ID:Station-Location">
<rdf:type><rdf:Description  rdf:about="http://www.w3.org/2000/01/rdf-schema#Class"/></rdf:type>
<rdfs:subClassOf><rdf:Description rdf:about="ID:User-Dynamic-
        Context"/></rdfs:subClassOf></rdf:Description>
<rdf:Description rdf:about="ID:Station-name">
<rdf:type><rdf:Description  rdf:about="http://www.w3.org/2000/01/rdf-schema#Class"/></rdf:type>
<rdfs:subClassOf><rdf:Description rdf:about="ID:User-Dynamic-
        Context"/></rdfs:subClassOf></rdf:Description>
<rdf:Description rdf:about="ID:Reviews-and-services">
<rdf:type><rdf:Description  rdf:about="http://www.w3.org/2000/01/rdf-schema#Class"/></rdf:type>
<rdfs:subClassOf><rdf:Description rdf:about="ID:User-Dynamic-
        Context"/></rdfs:subClassOf></rdf:Description>
<rdf:Description rdf:about="ID:location">
<rdf:type><rdf:Description rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-
        ns#Property"/></rdf:type>
<rdfs:domain>    <rdf:Description rdf:about="ID:Station-
        Location"/></rdfs:domain></rdf:Description>
<rdf:Description rdf:about="ID:category">
<rdf:type><rdf:Description rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-
        ns#Property"/></rdf:type>
<rdfs:domain>    <rdf:Description rdf:about="ID:Categories"/></rdfs:domain></rdf:Description>
<rdf:Description rdf:about="ID:reviews">
<rdf:type><rdf:Description rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-
        ns#Property"/></rdf:type>
<rdfs:domain>    <rdf:Description rdf:about="ID:Reviews-and-
        services"/></rdfs:domain></rdf:Description>
<rdf:Description rdf:about="ID:Categories">
<rdf:type><rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/></rdf:type>
<rdfs:subClassOf><rdf:Description rdf:about="ID:User-Dynamic-Context"/></rdfs:subClassOf>
</rdf:Description>
</rdf:RDF>
```