Department of Electrical Engineering and Information Technology
Distributed Multimodal Information Processing Group
Prof. Dr. Matthias Kranz

Development of an Android Driver Assistance System Based on V2X-Messages and Central Traffic Services

Entwicklung eines Fahrerassistenzsystems für Android basierend auf V2X-Nachrichten und Zentralen Verkehrsdiensten

Peter Christian Abeling

Master Thesis

Author:	Peter Christian Abeling
Address:	[redacted]
Matriculation Number:	[redacted]
Professor:	Prof. Dr. Matthias Kranz
Advisor:	Dipl.-Ing. Stefan Diewald
Begin:	01.10.2011
End:	31.03.2012

Department of Electrical Engineering and Information Technology
Distributed Multimodal Information Processing Group
Prof. Dr. Matthias Kranz

Declaration

I declare under penalty of perjury that I wrote this Master Thesis entitled

Development of an Android Driver Assistance System Based on V2X-Messages and Central Traffic Services

Entwicklung eines Fahrerassistenzsystems für Android basierend auf V2X-Nachrichten und Zentralen Verkehrsdiensten

by myself and that I used no other than the specified sources and tools.

Munich, March 16, 2012

Peter Christian Abeling

Peter Christian Abeling
Zaubzerstraße 45
81677 Munich

Abstract

Previous and current research on Vehicle-to-X (V2X) communication often focuses on system aspects, such as radio communications, networking/routing or information generation. This thesis tries to close the gap between technology prerequisites on the one hand and the requirements of potential users on the other hand. It aims at discussing design and implementation issues concerning selection and presentation of traffic information, as well as creating a consistent user interface and specifying a desired functionality from a potential user's point of view. By adopting a user-centred perspective, the focus is shifted to analysis of user acceptance, usability and the potential of V2X-based driver assistance. As integrating mobile phones into vehicles is an ongoing development, a driver assistance system prototype was designed, implemented and evaluated which is based on Android. The Android platform is an affordable and highly available alternative compared to an integrated solution. The system visualises traffic information from V2X communications and other traffic information sources such as Central Traffic Services (CTSs). Regarding the goal of the thesis, functional and usability-oriented requirements have been defined in order to guarantee a seamless integration into the in-vehicle context. Relying on a certain traffic model, different algorithms for managing and presenting warnings to the user have been developed for the current implementation. The modular software structure of the application allows developers to further expand the system's functions in the future.

Performance analysis results of the application showed that decoding incoming messages and rendering the integrated map view are particularly time consuming tasks. The warning management algorithms have been proven to be robust and efficient for the tested traffic scenarios. In order to get an impression of how potential users would interact with the system, a laboratory user study has been designed, executed and evaluated. The results show that the implemented driver assistance system convinced potential users of the increased traffic safety by demonstrating the potential of V2X communications. CTSs provide useful information which increase information density and availability if used in combination with V2X-based information sources. The benefit of the application can be further improved by integrating a navigation function in the system. Deploying mobile devices for the implementation of V2X communication services proved to be a considerable alternative given that all driver assistance tasks, such as navigation, traffic information and entertainment, are subsumed in one single system.

Kurzfassung

Bisherige und aktuelle Forschung im Bereich Vehicle-to-X (V2X) Kommunikation konzentriert sich oftmals auf die Felder Funkübertragung, Netzwerke/Routing und Generierung der Verkehrsmeldungen. Diese Arbeit versucht die Lücke zwischen technologischen Anforderungen auf der einen Seite und den Bedürfnissen potentieller Nutzer auf der anderen Seite zu schließen. Sie zielt darauf ab, aus der Sicht der Nutzer bestehende Design- und Implementierungsfragen bezüglich der Auswahl und Darstellung von Verkehrsinformationen zu diskutieren. Außerdem wird der Aufbau einer geeigneten Benutzeroberfläche, sowie die Spezifikation des Funktionsspektrums thematisiert. Da die Integration mobiler Endgeräte ins Fahrzeug eine fortschreitende Entwicklung darstellt, wurde im Rahmen dieser Arbeit ein Prototyp eines Fahrerassistenzsystems konzipiert, implementiert und getestet, welcher auf Android basiert. Die Android Plattform wurde hierbei als günstige und weitläufig verfügbare Alternative ausgewählt. Das System kann Verkehrsinformationen, die durch V2X Kommunikation gemeldet werden, visualisieren, sowie Informationen von anderen Quellen, wie zentralen Verkehrsdiensten, darstellen. Sowohl funktionale Anforderungen als auch Anforderungen an die Benutzbarkeit (*usability*) wurden definiert, um eine reibungslose Integration des Systems in den Fahrzeugkontext zu gewährleisten. Die modulare Struktur der Software wird im Detail erläutert, um Entwicklern die Möglichkeit zu geben das System in zukünftigen Arbeiten zu erweitern. Verschiedene Algorithmen, welche für das Management und die Anzeige der Warnmeldungen verwendet werden, wurden entwickelt, wobei ein spezielles Verkehrsmodel vorausgesetzt wurde.

Die Ergebnisse verschiedener Performance-Analysen ergaben, dass das Dekodieren empfangener Nachrichten und das Rendern der integrierten Kartenansicht besonders rechenintensive Aufgaben darstellen. Die Algorithmen für das Warnmanagement stellten sich im Falle der getesteten Verkehrsszenarien als robust und effizient heraus. Um einen Eindruck davon zu gewinnen, wie potenzielle Benutzer mit dem System umgehen werden, wurde ein Benutzertest erstellt, durchgeführt und ausgewertet. Die Ergebnisse zeigen, dass das entwickelte Fahrerassistenzsystem potenzielle Nutzer von einer größeren Verkehrssicherheit überzeugt, indem das Potenzial von V2X Kommunikation demonstriert wird. Zentrale Verkehrsdienste sind zudem sinnvoll, um die Informationsdichte und -verfügbarkeit von Warnmeldungen zu erhöhen, wenn sie in Kombination mit V2X Kommunikation eingesetzt werden. Der Nutzen der Anwendung kann darüber hinaus weiter gesteigert werden, wenn eine Navigationsfunktion in das System integriert werden würde. Die Nutzung einer mobilen Plattform für ein Fahrerassistenzsystem stellte sich als vorteilhafte Alternative gegenüber einer integrierten Lösung heraus, vorausgesetzt dass alle den Fahrer assistierenden Aufgaben, wie Navigation, Verkehrsinformationen und Entertainment in einer Anwendung vereint werden.

Contents

Chapter 1.

Introduction

A central goal of V2X communications is to warn drivers about situations in which traffic participant's safety is endangered. While the V2X communication network is responsible for generating and distributing data, the information needs to be received, decoded and analysed in each separate vehicle in order to present it to the user. An efficient visualisation of traffic information is crucial for successfully exploiting the benefits of V2X communications. Only if useful information is extracted and presented in an appropriate way considering the in-car context, users can be convinced of the advantages of V2X communication. Persuasion of users is in turn essential for a successful market introduction of the technology.

Since the availability of powerful mobile devices, such as smartphones and tablet Personal Computers (PCs), has increased[1], personal user devices are a considerable alternative to vehicle integrated systems. These devices are equipped with all means of communication, such as Wireless Local Area Network (WLAN), Bluetooth etc. which can be used for information transfer. In practice they can be personalised by the user and updated with low effort. Due to their availability on the mass market, the costs for the consumer are notably smaller than for a vehicle integrated system.

In the framework of this thesis, a driver assistance system prototype has been developed for the Android platform in form of an Application (App). The goal was to design and implement concepts for efficient visualisation of traffic information from V2X communication in order to identify user requirements for the system. The visual information system is used to verify concepts for V2X communication from the user's point of view. It is analysed which traffic information is useful or essential to be displayed in specific situations. Moreover, the Human Machine Interface (HMI) design is evaluated. The prototype can thus be used to increase the user acceptance for V2X communication because its benefit is visible in real scenarios. Additionally, the system can be modified for further research in this field. Researchers and manufacturers can use it to demonstrate the potential of V2X communications.

[1]Worldwide Mobile Communications Device Open OS Sales to End Users, Gartner Inc., `http://www.gartner.com/it/page.jsp?id=1622614`, last visited 3 March 2012.

Furthermore, other sources of traffic information have been included. Central Traffic Services (CTSs) collect data for a region and can be accessed via the Internet. Enriching V2X information with data from other sources is analysed as a solution to increase information density, especially when the V2X technology is not fully available.

The following goals have been defined for the prototype implementation:

- The Android application should be able to receive V2X messages and to decode and analyse them subsequently.
- Background services should be used to guarantee full warning support for situations in which the device is used for other tasks (e.g. the co-driver is surfing the web).
- A User Interface (UI) should be implemented which allows manipulating the system and its features.
- Two different methods of presenting the traffic information to the user are desired to be integrated.
- It should be possible to request and display traffic information from other sources, i.e. CTSs should be included.
- A database for storing incoming messages should further enhance the functionality of the system.

The focus has been set on user friendliness, intuitive handling and warning comprehension. The system was tested by simulating multiple traffic scenarios during development and in a final laboratory user test.

1.1. Related Work

The idea of using smartphones or tablet PCs in vehicles for analysing driving related scenarios is being researched extensively. The obtained data can, for example, be used to build up databases containing position-based information about road conditions. Mednis et al. propose a system to detect potholes using the mobile device accelerometer [1]. Analysing real world data, the authors achieved a true positive rate as high as 90 % which illustrates the potential of mobile device sensing. Another approach presented by Zaldivar et al. used an Android App running on a mobile device to monitor the vehicle's state by using the On Board Diagnostics II (OBD-II) interface [2]. Accidents are detected by analysing the G-force experienced by the passengers and the airbag triggers of the car. Internal vehicle data, together with data generated by the mobile device, is used simultaneously to achieve better accident detection rates. The information from both approaches could also be used to automatically generate V2X messages which alert a hazardous location (e.g. a pothole) or an accident.

For applications which analyse the driving related scenario in order to assist the driver, access to in-car context information is highly important [3]. As the data cannot be read directly from the vehicle's Controller Area Network (CAN) matrix, which is due to restrictions by the manufacturers, other approaches have been developed to access context information. Kranz et al. presented a concept for an interface which allows accessing vehicle data without the need to acquire the CAN matrix [4]. In their approach, sensor information is obtained from three different sources which are the standardised vehicle diagnostic interface OBD-II, a general purpose data interface reading and converting analogue signal wires, and a visual data recognition interface based on the diagnostic screen of the vehicle. In a next step, the collected data can be aggregated and interpreted to derive high-level context information. These high-level information could again be used to automatically generate V2X messages, as described in the previous paragraph.

Integrating mobile devices into the vehicle is another field of research relevant for this thesis. Diewald et al. showed how mobile devices can be integrated in the automotive domain in order to contribute to a Natural User Interface (NUI) experience [5]. A natural interaction with mobile devices in a vehicle is essential for a seamless integration and increases user acceptance and safety of usage. Using mobile devices in a vehicle requires modalities that allow the user to concurrently focus on the driving task. The authors claim that mobile devices represent a useful alternative for an In-Vehicle Infotainment System (IVI) because users are already familiar with manipulating their device regularly.

A similar work to this thesis was published by Grimm [6]. The author developed a platform to host several applications using V2X communication. A smartphone was used to develop new services without changes to the vehicle architecture. For receiving V2X messages, the Dedicated Short Range Communication (DRSC) gateway was used. His work demonstrates that the smartphone-centred approach is viable. Compared to this thesis, the user interface played a minor role in his work.

1.2. Thesis Structure

In Chapter 2, the basic principles of V2X communication are presented and motivated. Some use cases which are supported by the application are defined. At the end of this chapter, the designated hardware setup is explained, including details about how the system is used in real vehicles.

Chapter 3 briefly summarises the techniques used for providing CTSs and presents some common information sources.

A brief introduction to the Android platform is given in Chapter 4. The chapter covers the system architecture of an Android application, its framework and lifecycle. Finally, the programming tools used for this thesis are introduced.

The implementation of the driver assistance system is explained in Chapter 5. First, the requirements concerning usability and functionality of the App are defined. Second, all components which compose the App are explained. Third, their concepts and implementation particularities are commented and the modularity of the system is explained.

In Chapter 6, the simulation of traffic scenarios is presented. The simulations were used to test the application. The simulation tool *v2xMessageTester* is demonstrated, and some performance related topics are covered.

A laboratory user test was designed, executed and evaluated to test the application with the help of test users. All topics related to this test are explained in Chapter 7. Test results are interpreted and discussed at the end.

Chapter 8 encompasses a conclusion of test results and provides an outlook on possible future developments in this field of research.

Chapter 2.

Vehicle-To-X Communication

Vehicle-to-X (V2X) communication offers a high potential for improving traffic safety in the future. As modern cars are highly integrated systems equipped with several sensors, processing units and communication buses, it proves convenient to use information already present in the car in order to improve traffic safety and driver situation awareness. Exchanging information with other cars and V2X infrastructure grants in-car applications access to information of surrounding vehicles and decentralized information sources.

All information for the proposed driver information system essentially derive from communication between vehicles and between vehicles and infrastructure networks. The first variant is referred to as Vehicle-to-Vehicle (V2V), the other as Vehicle-to-Infrastructure (V2I). For abbreviation, the term V2X is commonly used. Lübke [7] defines four different global operational areas for V2X communications:

- **Vehicle-to-Vehicle Personal Communication:** Telephony, short messages, chat, etc.
- **Vehicle-to-Vehicle Traffic Safety Communication:** Brake warnings, emergency calls, traffic congestion warnings, convoy driving organisation, etc.
- **Vehicle-to-Infrastructure Personal Communication:** Data downloads, location-based services, car-park payment management, etc.
- **Vehicle-to-Infrastructure Traffic Safety Communication:** Communication with traffic lights, construction sites or road signs, traffic information downloads, weather information downloads, vehicle diagnosis, etc.

This thesis focuses on traffic safety communication because the presented driver assistance system aims at informing the driver about road safety issues. Personal communication is not within the scope of the thesis.

In the beginning of this chapter, some basic principles and challenges of V2X communication are explained. The second part explores the potential of general use cases. Afterwards, the Day-1 Use Cases defined for this thesis and supported by the corresponding Android application are illustrated. In the last part, the hardware setup of the application within the car is shown. It is explained how the information is retrieved technically and how the Android device is used in the car.

2.1. Basic Principles

V2X communication is a complex technology with different challenges to overcome (see Section 2.1.2). In order to create universal standards for the research world and industry projects, some car manufactures, supplier companies and research institutions established the Car-to-Car-Communication Consortium (C2CCC)[1] in 2002. The consortium published a manifesto in 2007 which describes the main building blocks of a V2X communication system [8]. This document was used as a reference for this section. Other sources are referenced where they occur.

As the standardisation progress has started some years ago, several specifications are already released by the European Telecommunications Standards Institute (ETSI). These standards play an important role in this thesis and in the V2X development process in general because they provide guidelines for research activities. Therefore they are reviewed in Section 2.1.3 separately.

2.1.1. Domains of a V2X Communication System

In their manifesto the C2CCC declares three different domains of a V2X communication system: The in-vehicle domain, the ad-hoc domain, and the infrastructure domain. Figure 2.1 shows the draft architecture of a V2X communication system [8].

The In-Vehicle Domain

The in-vehicle domain is represented by an On-Board Unit (OBU) and one or more Application Units (AUs). The OBU covers the in- and outgoing communication of the vehicle, whereas the AU runs an application using V2X data. The AU can be an integrated part of the vehicle or a portable device like a laptop or a smartphone. Both units can also reside in a single physical unit. The communication between both units can be wireless (IEEE 802.11a/b/g/n, Bluetooth) or wired.

The presented driver assistance system can be seen as a part of the in-vehicle domain. More precisely, an AU has been implemented by using an Android device and a dedicated driver assistance application running on it while the information is based on V2X communication. More details are given in Section 2.3.

[1]Car-to-Car Communication Consortium, `http://www.car-to-car.org/`, last visited 27 February 2012.

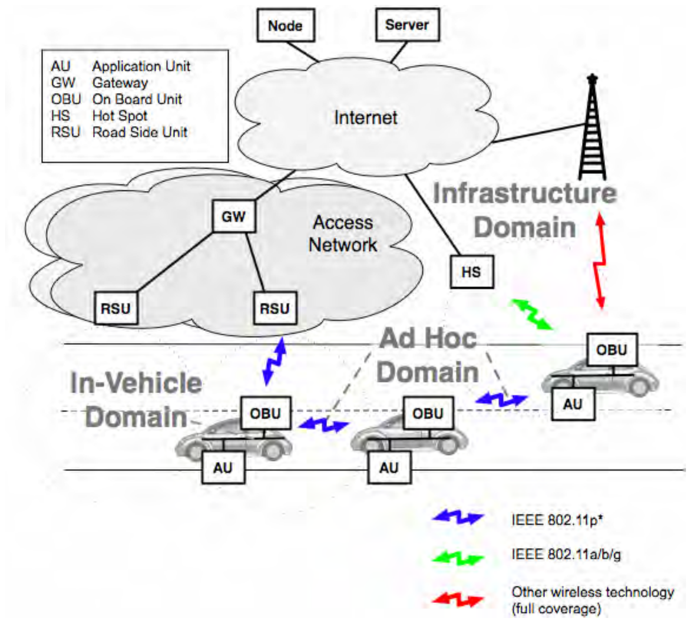


Figure 2.1.: V2X system architecture showing the infrastructure domain, the in-vehicle domain, and the ad-hoc domain[8].

The Ad-hoc Domain or Vehicular Ad-hoc Networks

The ad-hoc domain (or the Vehicular Ad-hoc Network (VANET)) consists of vehicles equipped with OBUs and Roadside Units (RSUs) which are placed along the streets. The OBU implements a wireless short range communication device. Without the need of a central coordination instance, OBUs form ad-hoc networks and communicate directly following dedicated protocols. If no direct connectivity exists between the sender and receiver OBU, special routing protocols allow multi-hop communications. This allows forwarding data packets from one OBU to another, until they reach their destinations. An example for such a V2V communication protocol is presented by Yang et al. [9].

Due to coverage difficulties, RSUs are needed to extend the range of ad-hoc networks and to enrich the network with additional data coming from other sources, such as the Internet or central traffic information entities. The RSUs are placed alongside the road and can be seen as static nodes of the ad-hoc network. As they are connected to an access network, RSUs enable OBUs to communicate with the "non-mobile" world.

As shown in Figure 2.1, an OBU can also communicate with a public or private Hot Spot (HS) to access Internet nodes or servers. These servers can be operated at home or by Internet service providers.

Organising VANETs poses a big challenge in the field of V2X communication. These challenges are further discussed in Section 2.1.2.

The Infrastructure Domain

The infrastructure domain composes the structural network behind the RSUs and HSs which are part of the infrastructure themselves. In case that an OBU cannot access an RSU to connect to the Internet, other wireless technologies are available (e.g. Global System for Mobile Communications (GSM), General Packet Radio Service (GPRS), Universal Mobile Telecommunications System (UMTS), High Speed Downlink Packet Access (HSDPA), Worldwide Interoperability for Microwave Access (WiMAX) and 4th Generation Mobile Telecommunications (4G)).

The infrastructure domain is also responsible for security and certification tasks. A Public Key Infrastructure (PKI) certification system is connected to distribute certificates used in wireless ad-hoc networks to ensure a specified level of security.

2.1.2. Challenges of V2X Communication

When reviewing different research papers in the field of V2X communication, it becomes obvious that several technical and non-technical challenges have to be resolved to successfully integrate the technology into the market and to create and increase user acceptance. This thesis contributes to the development, as the driver assistance system prototype was developed with a focus on the user interface and hence the user experience.

To summarize the challenges, Lübke [7] gives an adequate overview. They can be classified into three different groups.

Creation of a Robust Radio Infrastructure and Network

Generally, it is a technical challenge to use one single radio system for all operational areas of V2X communication (see introduction of Chapter 2). All services have different requirements due to performance, availability and privacy, hence an efficient network protocol with priority mechanisms is required. Furthermore, a high reliability and quality of service has to be achieved in order to assure the correct functionality of safety critical applications.

One of the most critical aspects is the need for an efficient scalable protocol. The number of users in dynamic VANETs changes constantly and quickly due to traffic volume and context. The distributed nature of vehicle-to-vehicle networks also requires multi-hop communication because the visibility of every network node cannot be guaranteed for all the time. This increases the protocol and routing complexity. In real-world situations, a high number of messages will be generated which creates a high network load. A reasonable trade-off between the latency of (time critical) messages and the number of participating users (i.e. the communication range) for one transmission has to be found. Yang et al. [9] propose a protocol to overcome these difficulties while exploiting congestion control policies. Their performance evaluations show promising results for future research.

Up to now, no radio standard could be agreed on for V2X communication. All the same, the Wireless Local Area Network (WLAN) Quality of Service (QoS) standard IEEE 802.11e can be one solution to solve the required issues [10].

Eventually, addressing network nodes (users) is a further challenge to cope with. In a constantly changing network topology routing tables run out of date quickly. At the same time, a message may not only be designed for one user, but for several users in a specific geographic area (e.g the message that a traffic congestion is ahead). This constraint leads to another routing mechanism called Geo-Routing. The idea of Geo-Routing is a position-based multi-hop forwarding of emitted messages. A message is forwarded in the geographical direction of its destination. A feasibility study and implementation of a Geo-Routing protocol was done in the framework of the FleetNet project [11].

Security Verification of Received Messages and Privacy of Drivers

Nowadays, security and privacy are key requirements in nearly all smart objects and environments [12]. The same holds for V2X communication. Warning messages must reach the user with short delay, as every millisecond can make the difference between an accident or a safe stop (e.g. when the car in front is braking sharply). Therefore, the introduction of trust and trustworthy services will be crucial for the successful introduction of V2X communication. Again, a trade-off between message delay and time-consuming security mechanisms, such as cryptographic calculations, has to be found.

As users show more and more concern about location-tracking issues [13], privacy concerns might arise with respect to location dependent services in V2X communications. Privacy of users must be guaranteed, as well as data security and reliability.

Market Introduction and Preparation of Interesting Services

Only if V2X communication provides a clear and measurable improvement of driver security and offers a wide range of new services, it can be successfully integrated into the modern car market. “Early buyers” cannot use the system’s full capabilities, as approximately 10% technology penetration is needed to significantly improve driver security [14]. This increases the importance of a RSU network which can offer a wide range of services that can be used without a high number of OBUs.

2.1.3. Specifications

As research in the domain of V2X communication has started some years ago, the standardisation progress closely follows new developments. Different aspects of V2X communication need standards in order to provide a reliable and common basis for companies and researchers, including radio specifications, network/protocol specifications, message formats, application sets, etc.

Decisive for this thesis are a set of Technical Specifications (TS) released by the ETSI in recent years. Most important is the definition of the Basic Set of Applications for Intelligent Transportation Systems (ITS). Part one specifies the Functional Requirements of a V2X communication system [15]. Besides, a set of use cases is defined (see Section 2.2).

In part two, the Cooperative Awareness Basic Service is specified in detail [16]. This part includes services based on awareness of other vehicles around the user vehicle. The definition of the **Cooperative Awareness Message (CAM)** specified in the Abstract Syntax Notation One (ASN.1) notation [17] is integral.

Part three specifies the Decentralized Environmental Notification Basic Service [18]. This service includes communication with RSUs and HSs to exchange information. For this type of communication, a **Decentralized Environmental Notification Message (DENM)** is defined in this document in the ASN.1 notation.

CAMs and DENMs form an important part of V2X communication in general and are essential for this thesis because they carry V2X information. They are therefore explained in detail later in this section.

Alongside the named specifications, the ETSI released several other standards with respect to communication and network architecture, GeoNetworking etc. They are not referenced in this work but can be found with the aid of the ETSI Standards Search engine[2].

Co-operative Awareness Message (CAM)

Röckl et al. presented a concept of cooperative awareness for driver assistance systems [19]. CAMs are sent and received by all participating moving participants in a V2X scenario, such as private cars, emergency vehicles or public transport vehicles. The exact format of a CAM is specified in ETSI specification TS 102 637 Part 2 [16]. As V2X communication is still researched extensively, the exact specification may change in the future. The basic idea of CAM is to communicate a "Here I Am"-like information to all participating users in the communication range.

Figure 2.2 shows the root of the CAM structure as specified in the standard [16]. The root (*CAM Protocol Data Unit (PDU)*) contains a header and a child object named *Cooperative Awareness*.

[2]ETSI Publications Download Area, `http://pda.etsi.org/pda/queryform.asp`, last visited 27 February 2012.

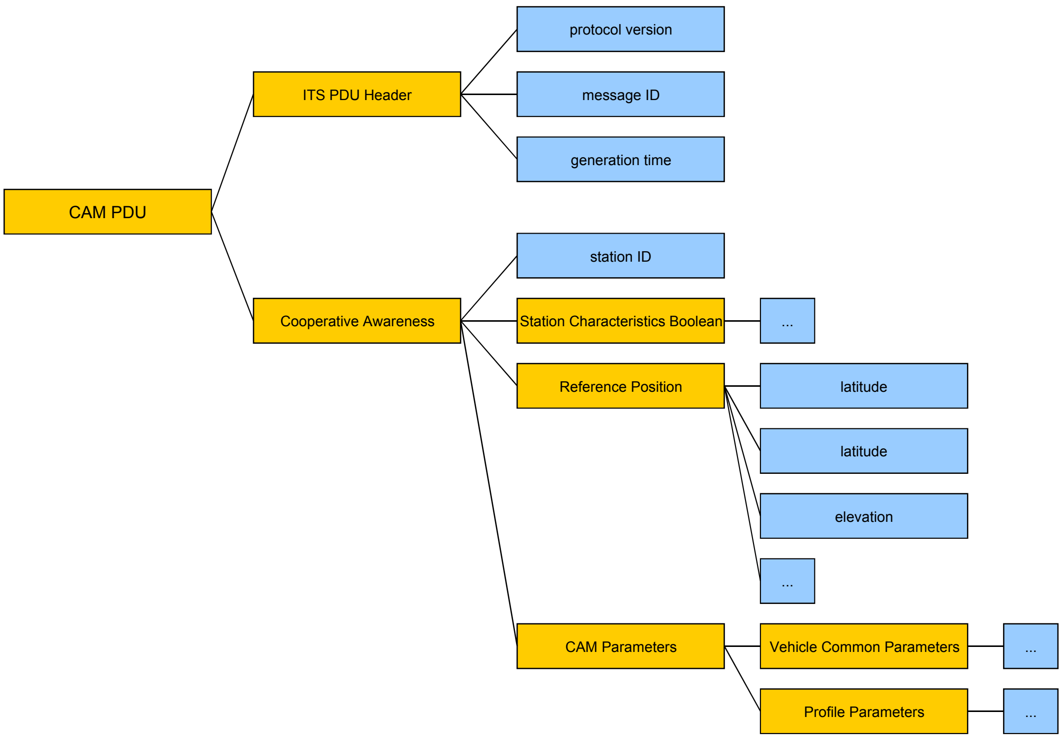


Figure 2.2.: CAM structure as specified in the standard [16]. Yellow boxes symbolise objects, blue boxes symbolise variables. Not explicitly mentioned variables are indicated by boxes containing dots.

The header stores some packet related information values. The child object contains (besides the station ID) three child objects related to simple station characteristics, as well as the current station position and other station parameters. The *Station Characteristics Boolean* object allows a quick determination of some station related boolean values, which indicate e.g. if a station is private, mobile or physically relevant. The *Reference Position* object contains all position-related variables. Not only the current latitude, longitude and elevation of the station are stored here, but also optional parameters about the driving direction, street name, road segment etc. The *CAM Parameter* object is constructed of common parameters for all vehicle types and of profile dependent parameters. The first object can store e.g. the width and length of the station, current speed, acceleration rate, and confidence intervals for some values. So far, there are three types of vehicle types specified: the basic vehicle, the emergency vehicle and the public transport vehicle. Dependent on which type is represented by the CAM, the *Profile Parameter*s contain information about type related aspects. An example is the information whether or not the siren and light-bar are in use.

CAMs will account for the bulk of V2X communication as they are emitted regularly with a frequency of $10\,\mathrm{Hz}$ from every moving participant. Encoding and decoding of CAMs therefore must be efficient and value access simple. Details about CAM related implementation issues can be found in Chapter 5. The ASN.1 description of CAM can be found in the standard [16].

Decentralized Environmental Notification Message (DENM)

DENMs are responsible for notifying drivers about an event related to their own situation. They are triggered and repeated when an event occurs. Triggering can be done by a central management unit or by a participating OBU or RSU. It is also possible that an event-initiator, such as a construction site, directly sends out DENMs. In this case, the construction site network node can be seen as an RSU. DENMs are broadcast as long as the event cause stays active. Details about repetition rate, broadcast radius and multi-hop issues are not yet fully clarified.

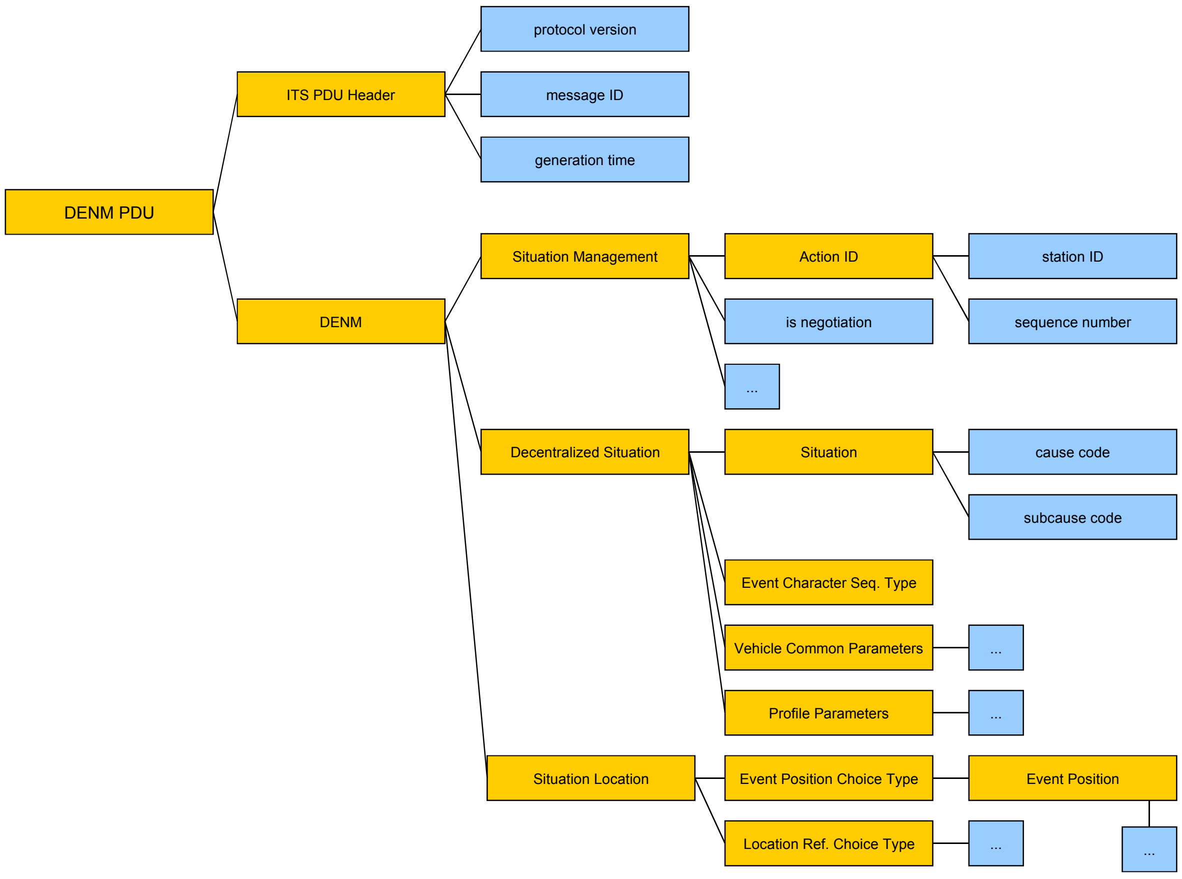


Figure 2.3.: DENM structure as specified in the standard [18]. Yellow boxes symbolise objects, blue boxes symbolise value variables. Not explicitly mentioned variables are indicated by boxes containing dots.

In Figure 2.3, the format of a DENM as specified in the standard [18] is shown. In comparison to CAM, DENM has more data fields as it carries more information. The header is the same as for CAM. DENMs consist of three child objects, a *Decentralized Situation Management* object, a *Decentralized Situation* object and a *Decentralized Situation Location* object.

The *Decentralized Situation Management* object carries the information about the station ID and sequence number. It also informs the receiver when an event is not active any more in adjusting the *is negotiation* variable. Other values from this class can define the expiry time, reliability information, or message frequency.

The *Decentralized Situation* object wraps information about the situation itself. The *cause code* and *subcause code* variables are used together with a look-up table to signal the event cause. The object *Event Character Sequence Type* contains information about the character of the event, e.g. if it is mobile, time critical or relevant at all. The other two child objects are the same as for the CAM message (see Figure 2.2).

The *Decentralized Situation Location* object implements detailed information about the event's location. As for a CAM, the final position is given as a latitude and longitude pair (not shown in the figure, part of the *Event Position* object). The last child object *Location Reference Choice Type* enables the user to track the route of the event by providing a collection of previous waypoints and a trace ID.

Details about DENM related implementation issues for this thesis can be found in Chapter 5. The ASN.1 description of DENM can be found in the standard [18].

2.2. Vehicle-To-X Use Cases

The collection of use cases for V2X messages is a strong argument for motivating its usage in modern automotive environments because it shows the potential of such a technique to increase traffic safety and driving convenience.

The first part gives an overview of general potentials of V2X communication with the help of an example. In the second part, Day-1 Use Cases which have been chosen for this thesis are presented and discussed.

2.2.1. General Potential for Traffic Safety

Up to now, most information about the surrounding of the driver is gathered by direct visual observation or sounds. Additionally, some modern navigation systems or in-car systems offer the possibility to get information about road conditions or traffic volume via radio or mobile phones. Their idea resembles the idea of V2X communication which is to overcome the limits of direct surrounding perception with the help of telecommunication.

As mentioned above, cars nowadays use a broad selection of sensors to gather information about their state and their environment. This information is not actively shared with other participants and is hence not available to others. The vision of V2X communication is to quickly pass on information from one car to another in order to extend driver awareness and perception [11].

An example from Yang et al. [9] shall be used to further clarify the case.

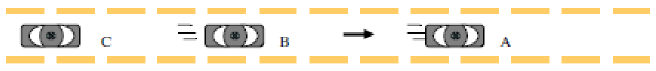


Figure 2.4.: V2X road safety improvement scenario [9]

Figure 2.4 shows a common road situation: three cars (A, B and C) are driving one after another with high speed. Suddenly A brakes abruptly. At that moment, both cars B and C are endangered to collide with the car in front of them. C does not have the advantage of being further away from the actually braking car A because its line of sight is limited by B. In a worst-case scenario, all drivers can only see the brake lights of the car in front of them. Many studies show that the reaction time of a driver between the perception of brake lights and stepping on the brake varies between 0.7 and 1.5 seconds [20]. Every single driver's reaction time therefore further increases the delay of the emergency brake message in the direction of following cars.

The use of V2X communication can significantly decrease the propagation delay of an emergency message. Using direct wireless communication between the cars in our example would have given B and C access to the information that A is braking almost immediately and simultaneously. C would know that B is going to brake sharply which enables him to initiate an appropriate action to prevent a collision.

This example demonstrates a huge advantage of V2X techniques: the perception of the driver is widened up to several hundreds of meters [9]. Drivers are enabled to "look" at road segments which are out of sight for the moment and around corners. Poor visual conditions can be compensated. Furthermore, their attention can be forced to an event with the aid of appropriate applications in the car which monitor incoming messages. The driver is informed in advance about approaching emergency vehicles or motorcycles, hazardous locations, emerging traffic congestions, accidents, stationary vehicles, bad weather conditions, roadworks etc.

When interpreting V2X message data, a wider range of applications is thinkable. Smart algorithms can determine driving directions and speed of different cars to warn about an imminent collision or sharp braking vehicles. Other applications aim at communicating with traffic lights or road signs to force the driver's attention to it in time.

2.2.2. Day-1 Use Cases

As many applications can be imagined and are currently under the scope of research, a set of use cases has been defined which is supported by the driver assistance system developed for this thesis. The set is called Day-1 Use Cases as it stands at the beginning of several possibilities which can be added later to the system.

The driver assistance system named *DriveAssist* is an Android application. Its functionality and implementation are explained in Chapter 5. The application is able to fully support the Day-1 Use Cases which use both CAMs and DENMs. During the test and simulation process only these cases were considered. Nevertheless, it is possible to extend the application's functional range. Details on how to obtain extensions are also provided in Chapter 5.

The following sections each cover one of the six Day-1 Use Cases. The cases are all part of ETSI specification TS 102 637-1 [15]. They are explained by their underlying idea and how they are represented by a message. Algorithmic issues, such as the treatment of multiple events at the same time, expiry times, priorities etc. are covered in Chapter 5. Moreover, all events are assumed to happen on the same lane where the user is driving. This allows avoiding lane management issues. Table 2.1 summarises the Day-1 Use Cases.

Day-1 Use Case	notified by
Approaching emergency vehicle	CAM
Electronic Emergency Brake Lights (EEBL)	DENM
Stationary vehicle	DENM
Traffic congestion	DENM
Roadworks	DENM
Hazardous location	DENM

Table 2.1.: Day-1 Use Cases supported by *DriveAssist*

Approaching Emergency Vehicle

Emergency vehicles include all vehicles on an urgent mission, like ambulances, fire engines or police cars. When an emergency vehicle approaches, drivers need to know the exact distance and direction from which it is coming in order to be able to make room if necessary. A CAM with the variable *vehicle type* set to "emergency vehicle" is used to indicate an approaching emergency vehicle via multi-hop communication to other traffic participants (Figure 2.5). The specification also declares that a DENM can be constructed in complementary of CAM, but how this is done or what the role of this DENM will be has not been specified yet. Therefore *DriveAssist* will only support CAMs for this use case which is sufficient.

Thus, *DriveAssist* receives CAMs which are sent out by the emergency vehicle itself or which are forwarded by other participants. From the viewpoint of the receiver both possibilities do not make a difference, only the range increases when using multi-hop techniques. In simulated environments, such as used for this thesis, the range was always chosen as being sufficiently big enough without the need of multi-hop forwarding.

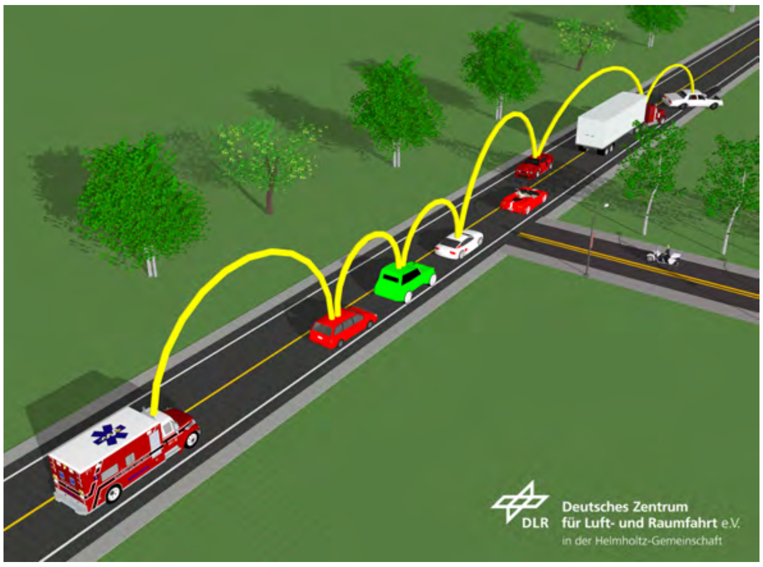


Figure 2.5.: An approaching emergency vehicle announces its presence to other traffic participants via multi-hop vehicle-to-vehicle communication using a CAM.

The driver will be able to follow the emergency vehicle from its entry to its exit of the chosen communication range radius around the car. The application takes care of displaying the situation in an appropriate way.

Electronic Emergency Brake Lights (EEBL)

The EEBL warning aims at improving the situation shown in the chosen example in Figure 2.4. Two variants of determining a sharp braking vehicle are possible from the receiver point of view:

Firstly, the braking vehicle can generate a DENM itself and broadcast it when on-board sensors detect sharp braking activity. Secondly, every receiver checks incoming CAMs from other vehicles for rapidly changing speed occurrences. Together with the position of the braking car, algorithms can determine if a vehicle is braking and broadcast a DENM with an appropriate warning.

Comparing the two approaches shows advantages and disadvantages for both techniques. The first approach adds complexity to the sensory system (and therefore a small calculation delay) to the braking car. Furthermore, the information that the car is braking is only determined once from one source. The latter can also serve as an advantage, as the message comes from the source and is trustworthy if determined correctly. More direct information than the speed is used (e.g. the stepping on the brake pedal). The second approach adds complexity to the AU of every participant. Only the vehicle speed and acceleration can be used to identify braking activity. Moreover, at least two consecutive CAMs are needed which adds another undesirable delay.

How exactly this case will be dealt with in reality is not specified by the ETSI at the moment.

For this thesis, it is assumed, that every vehicle detects its own braking activity and broadcasts an appropriate DENM to announce it. A special *cause code* is defined for an EEBL event. Multi-hop forwarding can be used (also via RSUs) to increase the communication range.

DriveAssist will display braking events around the vehicle to the driver.

Stationary Vehicle

A stationary vehicle is a vehicle stopping on the road, the emergency lane or physically relevant close to the road. The reason can be an accident, a technical, or a personal problem . Similar to the EEBL, either the stationary vehicle itself or receivers could determine the status "stationary". It is also possible that a third, central authority detects stationary vehicles (via the RSU network) and broadcasts DENMs. Such a DENM can be seen as an "electronic warning triangle".

In this thesis it is assumed that the receiver is not itself responsible to detect stationary vehicles, but is explicitly informed about it by a special received DENM *cause code*. Whether the message comes from the stationary vehicle itself, from other participants via multi-hop or from central authorities is subsidiary.

The ETSI specifications provide a finer distinction of stationary vehicles, e.g. whether it is caused by an accident or a technical problem. This differentiation is done by using different *subcause codes*.

The DENM will contain position information about the location of the event. *DriveAssist* will control incoming DENMs to detect stationary vehicles and signalise them to the driver.

Traffic Congestion

Traffic congestions are the most common traffic events. Automatic detection of a traffic congestion can, for example, be done by analysing the speed and position of several cars next to each other. If they drive close by, relatively slow or not at all and if these events occur in an adequate number, a traffic congestion is very probable.

It is envisioned to enable every participant to detect a traffic congestion on their own [15]. This thesis again relies on a central instance or participant sending DENMs with a defined *cause code*. The location information included in the message points to the closest part of the traffic congestion which normally is the current last car. The application *DriveAssist* is able to display traffic congestions to the user.

Roadworks

Roadworks or construction sites are very likely to be treated as an RSU in the future. They will send out DENMs constantly to announce their presence. *DriveAssist* is able to detect roadworks and display them to the user.

Hazardous Location

A hazardous location is an abstract description for all location-based dangers, such as ice on the road, bad weather conditions, obstacles, oil or grit on the road or bad road conditions. Similar to the stationary vehicle case, the DENM *subcause code* can be used to specify the respective case.

In regard to the broadcasting scheme, *DriveAssist* relies on a central entity announcing hazardous locations, as it is not yet decided who will be responsible for monitoring hazardous locations in the future.

Nevertheless, *DriveAssist* is capable of informing the driver about hazardous locations and their positions and characteristics in an appropriate way.

2.3. Driver Assistance Hardware Setup

V2X communication is still part of many different research projects. Diverse challenges are to master, not less investors and potential clients must be convinced of a significantly increased driver safety and comfort (see Section 2.1.2). The driver assistance system designed, implemented and tested in this thesis aims at contributing to these developments. In providing a fully developed graphical user interface to access V2X information, clients can directly experience real or simulated scenarios from the driver's point of view.

In order to keep the implementation flexible and cheap, it was chosen to use an Android-based device (e.g. a smartphone or tablet Personal Computer (PC)) with Android running on it as a platform to develop a driver assistance system (see Chapter 4 for details about the Android platform). This approach avoids a more complex and more expensive integrated solution, but enables researchers, developers and clients to access the full range of V2X capabilities if they are supported by the application.

During the application development, no real V2X test facilities were available. These kinds of tests are elaborate and time-consuming, as different vehicles and infrastructure equipped with V2X technology need to be provided. Instead of real in-vehicle tests, a simulation environment was used which simulates the emission of V2X messages. Nevertheless, the developed application is designated to be used in real V2X scenarios later. The simulation environment is presented in Section 6.1.

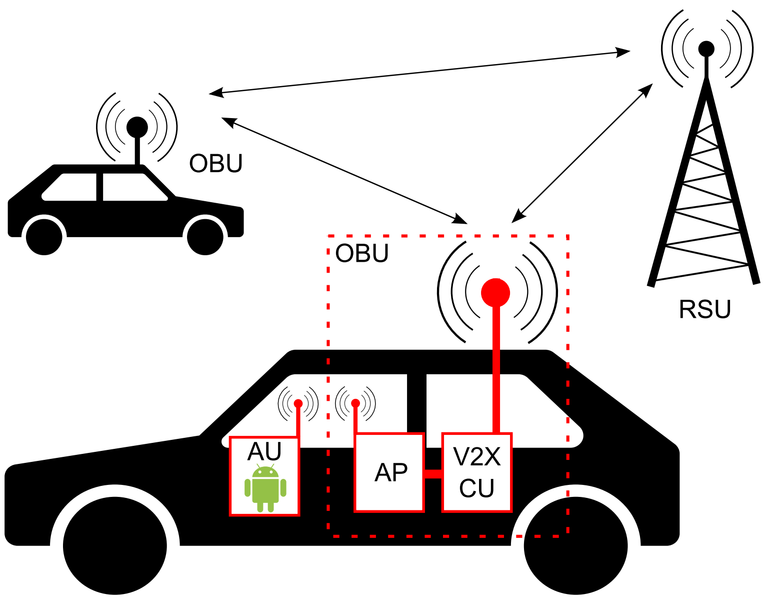


Figure 2.6.: Designated V2X hardware setup and communication links between OBUs and RSUs. Inside the vehicle, the AU receives traffic information received by the V2X CU via an AP.

Figure 2.6 shows the hardware setup for the scenario. The car in the front is exemplary equipped with an OBU and an AU. All moving participants need at least an OBU to exchange information via V2X communication. The OBU communicates with other OBUs and RSUs at the roadside (see Section 2.1.1).

An OBU consists of a V2X CU, an AP and a V2X antenna. AP and CU are connected or can even be one single physical unit. The CU is responsible for sending and receiving V2X messages, as well as for encoding and decoding them. It is also possible that the CU implements filtering algorithms for preselecting or dismissing messages. Moreover, the CU can access vehicle communication buses, such as Controller Area Network (CAN), for monitoring sensors or vehicle states. Communication with GSM, UMTS or 4G infrastructure can also be realised with the CU in using the antenna of the vehicle.

The AP provides a wireless in-vehicle network with local addresses. One or more application units can access the network to receive decoded messages from the CU. In this thesis, the AU was chosen to be an Android device. The device can be mounted in the car with a holder in order to be visible to the driver for the whole time.

Chapter 3.

Central Traffic Services

A Central Traffic Service (CTS) is a source of current traffic information for a certain region (for example a German Federal State). The available information is collected and aggregated by a service provider. Many different means of communication are exploited for distributing the data in today's systems, such as Very High Frequency (VHF) radio using the Radio Data System (RDS), Global System for Mobile Communications (GSM)/General Packet Radio Service (GPRS) or Universal Mobile Telecommunications System (UMTS)/High Speed Downlink Packet Access (HSDPA). Providing nearly real-time traffic information aims at informing the driver about traffic incidents in order to prevent accidents, save fuel and improve the capacity utilisation of the traffic network.

CTSs can use different information sources. Modern services use their own system for generating and redistributing traffic data. Common sources are:

- the police
- road maintenance staff
- private persons, radio stations and the German Automobile Club (ADAC)
- sensors (cameras, light barriers, induction loops)
- Floating Phone Data (FPD) (position analysis in network provider cells)
- Floating Car Data (FCD) (user position uploaded by user navigation systems and aggregated by service provider)

As automated sources, such as FPD and FCD, are particularly interesting due to their actuality, distribution, and potential, they are explained in detail in the upcoming sections.

3.1. Floating Car Data

Floating Car Data (FCD) is generated by aggregating position and movement information from every user with an appropriate device. For position measurement, a Global Positioning System (GPS)-online receiver obtains the current position and uploads it to the server of the provider. Those receivers can be navigation system devices or smartphones running an appropriate application (e.g. NAVIGON select[1]). The information is enriched by adding data from fleets with a high distribution on the roads, such as taxis, logistics or car rental companies.

New approaches add context information from the car to further increase the precision and value of the traffic information. Huber et al. present the eXtended Floating Car Data (XFCD) system developed by BMW and other partners [21]. The classic data set (position and speed) is extended by adding data from the vehicle's data bus. The information originates from the vehicle's sensors. Examples are the windscreen wipers or the rain sensor, the external thermometer, the vehicle's light system or the systems to control the vehicle dynamics, such as Anti-lock Braking System (ABS) or Electronic Stability Control (ESP). According to the authors, it is possible to gather additional high-context information about a traffic situation when using data concerning the road or weather conditions etc. Similar to standard FCD systems, all data is processed on-board and passed to an information center in form of situation or traffic messages.

3.2. Floating Phone Data

Floating Phone Data (FPD) is obtained by the mobile phone network. Mobile phone users serve as samples in order to derive time-space trajectories of car travellers, as it is presented by Friedrich et al. [22]. During phone calls, FPD can serve as an alternative to FCD services, being cheaper and broader. Even data sets collected from mobile phones in stand-by mode can be used to analyse route choice behaviour when sampled over longer time periods.

3.3. Information Sources

For scientific purposes CTSs such as data from *Antenne Bayern Stauservice* (GeoRSS), *Bayerischer Rundfunk "Staus und Behinderungen"*, *Google Maps* and *TomTom HD Traffic* have been used temporarily for the implementation. Exemplary for a CTS, *TomTom HD Traffic* is briefly reviewed in this section.

[1]NAVIGON select (Android market): `https://market.android.com/details?id=com.navigon.navigator_checkout_eu40&hl=de`, last visited 27 February 2012.

TomTom HD Traffic is based on FCD and FPD and additionally integrates Traffic Message Channel (TMC) data. FCD is obtained by TomTom LIVE navigation systems. For getting FPD in Germany, TomTom has contracted a collaboration with Vodafone to anonymously analyse mobile phone data from Vodafone clients. This includes the potential of 36,706 million mobile phone clients in Germany in 2010/2011[2]. TMC is used to enrich the data with longer lasting and planned events, such as construction sites or road blockings. Further details about the technical background can be found in the White Paper published by TomTom N.V. [23].

Technically, a proxy server from the Distributed Multimodal Information Processing Group (VMI) regularly requests current traffic information and provides it towards the application using JavaScript Object Notation (JSON) objects. Further details on the implementation are given in Section 5.2.4.

[2]Source: Vodafone Pressemitteilung from Mai 15, 2011, http://www.vodafone.de/unternehmen/presse/pm-archiv-2011_188584.html, last visited 27 February 2012.

Chapter 4.

Introduction to the Android Platform

Android is a software-platform and an operating system for mobile devices such as smartphones, mobile phones, netbooks, and tablet Personal Computers (PCs). It is developed by the Open Handset Alliance (OHA)[1] founded by Google Inc. and 33 partner companies. Nowadays, the OHA consists of a total number of 84 member companies, including hardware manufacturers, mobile carriers, and software developers. As an open-source software stack Android was published under the Apache Software License 2.0[2] and therefore enables the developer to access all hardware and software components of the device. The great success of Android is confirmed by several studies on mobile operating system usage, for example by the market research company Gartner[3]. Android's worldwide market share in mobile smartphone sales in the second quarter of 2011 has been estimated to 43.4 %, followed by Symbian (22.1 %) and Apples iOS (18.2 %) in this study.

In this chapter, a short introduction to the Android platform is given with respect to Android's system architecture and framework, application life-cycle and available programming tools. A special focus is set to further particular aspects which are relevant for this thesis. This chapter provides basic information about Android, but does not cover the application which was developed for the thesis in particular.

4.1. System Architecture

The Android software stack consists of several elements. It is shown in Figure 4.1.

For a consistent hardware abstraction, the lowest layer is composed of a **Linux kernel** optimised for mobile devices. This makes it possible to use Android on devices of different manufactures, as the drivers and power management are adapted to the particular device.

[1]Open Handset Alliance, `http://www.openhandsetalliance.com/`, last visited 27 February 2012.
[2]Android Open Source Project, `http://source.android.com/source/licenses.html`, last visited 27 February 2012.
[3]Worldwide Smartphone Sales to End Users by Operating System in 2Q11, `http://www.gartner.com/it/page.jsp?id=1764714`, last visited 27 February 2012.

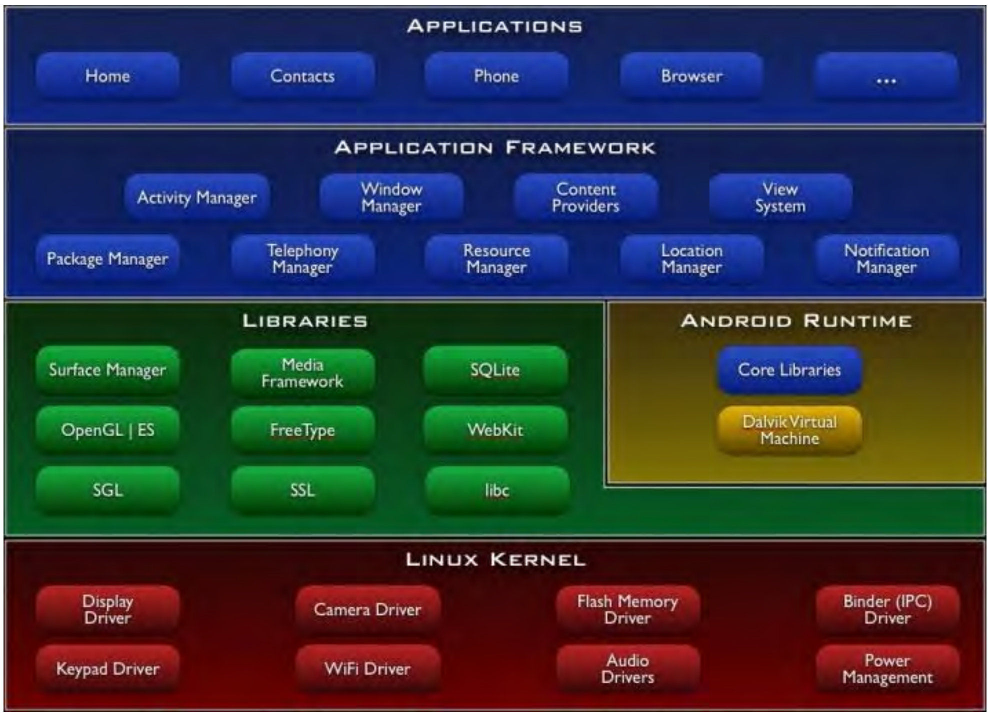


Figure 4.1.: Android software stack[4]

The overlying layer contains efficient C-based **libraries**, like OpenGL for 3D graphics and Android's built-in database system SQLite. These libraries define a basic set of functions for upper layers. Additionally, the **Android runtime** provides a Virtual Machine (VM) called Dalvik. As Dalvik uses his own data format, the high-level Java code is translated for the VM.

The **application framework** is a collection of helper classes for implementing Android applications. It also manages the abstracted hardware access for the application, the user interface and the application resources. A more detailed view on the framework is given in Section 4.2.

The top layer is the **application layer**. This layer is visible to the user and includes native applications from the manufacturer, applications from the Android market or own developments. All applications have the same hierarchy and are written in Java programming language[5]. The driver assistance system application *DriveAssist* is also located here.

[4]Android Architecture, `http://developer.android.com/guide/basics/what-is-android.html`, last visited 27 February 2012.

[5]The Native Development Kit (NDK) allows to program parts of the application in C/C++. The NDK was not used for this thesis.

4.2. Application Framework

The application developed for this thesis makes broad use of several different components of the application framework. This section briefly introduces the basic components available to developers, to give the reader a better understanding when the application is explained in Chapter 5.

The following elements provided by the framework are the architectural base for every Android application [24]:

- **Activity:** Activities are the particular screens presented to the user. Activities respond to user interaction and use Views to form graphical user interfaces.
- **Service:** Services run in background threads and therefore do not need user interaction. They perform regular processing, even when the applications' activities are not active or visible.
- **Intent:** Intents are used for communication between Activities and Services. Information can be sent to a particular destination or broadcast system-wide.
- **Broadcast Receiver:** A broadcast receiver listens for particular Intents and performs actions when it receives one. Broadcast Receivers are important for event-driven tasks.
- **Notifications:** Without interrupting the user while using the device, a Notification signals a certain event or finished action in the notification bar of the device. It can also inform users about running services on their devices.
- **Content Provider:** Content Providers can be used to access or publish data from or to other applications over an interface. Content providers are not used in this thesis.
- **Widget:** A widget is a visual component of an application which can be added to the home screen. Widgets are not used in this thesis.

From the programming point of view, these elements are derived from helper classes and are extended with the designated functionality. Afterwards, they can be combined modularly.

4.3. Application and Activity Lifecycle

An application basically consists of Activities and Services which are created and destroyed dynamically during runtime by user interaction. The other components are used to perform the starting and stopping of Activities and Services (Intent, Broadcast Receiver), inform users about occurred events (Notification), access special data (Content Provider) or provide a special graphical interface (Widget). Obviously, the application state is depending on what the user is currently doing, e.g. which functions of the application he/she is using.

Usually, when starting the application, the first Activity is displayed. From here, the user can actively navigate or is navigated to other Activities. Simultaneously, services can be started automatically or manually or Intents can be sent and received etc. For that reason, the application lifecycle is strongly depending on the desired functionality.

Each Activity within an application has a predefined lifecycle to let the system efficiently manage events like incoming calls/messages, pressing the home-button etc., which force an Activity to the background. It also enables the programmer to find predefined entry and exit points for each Activity.

The Activity lifecycle is shown in Figure 4.2.

When an Activity starts, its `onCreate()` method is always called. To arrive at the running state, the functions `onStart()` and `onResume()` are called one after the other. All three functions can be used to initialise the Activity. If the Activity looses focus, the function `onPause()` is called. When it comes to the front again, `onResume()` is called. It can be killed manually by the user or by the system. Another common case is that the application is no longer visible because the user starts another Activity or presses the "back"-button. In this case, the functions `onStop()` and `onDestroy()` are called. These functions should be used to release resources. All functions can be overridden and adapted to the programmer's needs.

4.4. Programming Tools and SDK

For the development of the application presented in this thesis, common Android development tools have been used [24]. These include:

- ***Eclipse* and Android Developer Tools (ADT) plug-in**: *Eclipse* is a powerful and easy-to-use programming environment. It is the preferred tool for Android developers, because a plug-in is available which integrates ADT, such as an emulator, into the Integrated Development Environment (IDE). Furthermore, it automatically creates the application and provides several debugging tools. For this thesis, the *Eclipse* version 3.7 ("Indigo") was used.

- **Java Development Kit (JDK):** As Android applications are programmed in Java, a version of the JDK has to be installed. For this thesis, the JDK 7 was used.

- **Android Software Development Kit (SDK):** The SDK provides the user with documentation, sample programs and several helpful tools, such as the *aapt* for managing application resources and the emulator. Different Application Programming Interface (API) levels have been released. In this thesis the API level 10 was used. The application is also compatible with all higher Android versions.

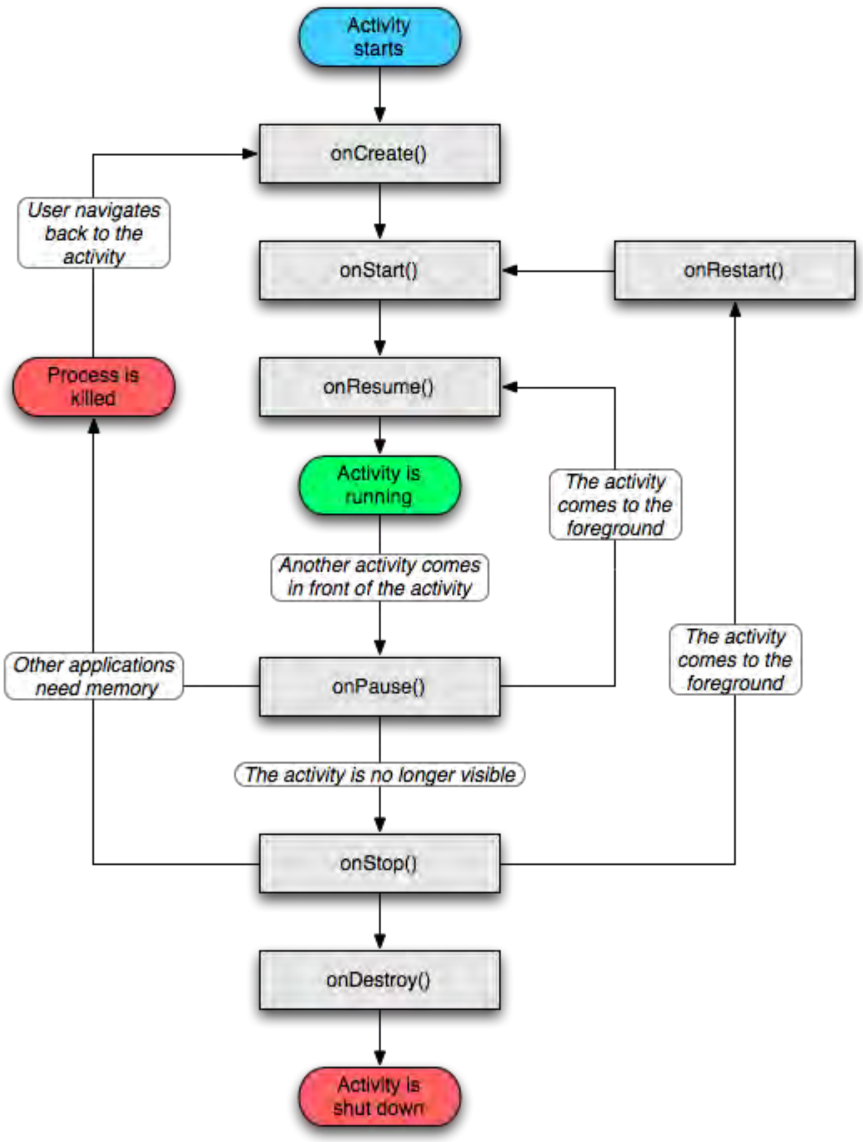

Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Figure 4.2.: Android Activity lifecycle[6]

- **Dalvik Debug Monitoring Service (DDMS):** The DDMS is a debugging interface between the IDE and the application. Processes on the emulator or an Android device can be analysed. It also allows debugging the source code by using breakpoints and analysing the application's timing behaviour.
- **Android Debug Bridge (ADB):** The ADB allows managing the emulator or the connected Android device from the terminal. It is especially interesting because the application can be

[6]Activities, `http://developer.android.com/guide/topics/fundamentals/activities.html`, last visited 27 February 2012.

directly installed on the device as an apk-file.

Android is an Open Source Project and therefore has an active developer community. Several forums exist where developers exchange support for complex programming tasks. Additionally, the Android developers homepage[7] provides a huge amount of information including design guidelines for graphical elements and best-practice tips.

[7]Android Developers, http://developer.android.com/index.html, last visited 27 February 2012.

Chapter 5.

Prototype Implementation

The driver assistance prototype *DriveAssist* was developed as an Android application. This chapter explains all details considering this application regarding its implementation. It also covers the requirements it has to meet in Section 5.1. The structure and components of the system are presented in Section 5.2.

5.1. Application Requirements

5.1.1. Usability requirements

Using Android as a platform for a driver assistance system is motivated by the requirements of the application. An Android-based visual information system offers the possibility to implement and test mechanisms to process and analyse traffic information from different sources, as well as to evaluate methods for presenting information to the user. Android devices are highly available and can be used in laboratory environments as well as in test vehicles. They are equipped with all established means of communication, especially WiFi (IEEE 802.11b/g/n) will be exploited here. No other hardware components are needed, and the high costs and complexity of developing an integrated solution can be avoided.

At the same time, the focus has to be set to the user interface. The application is designated to be used in cars and other vehicles and therefore has to fit special requirements. It should not demand attention from users while driving in order not to put driving safety at risk. McKnight et al. describe a study in which the effect of driver distraction by cellular phone calls and other activities, such as turning on the radio, was observed [25]. All distractions led to a significant decrease of driver attention, especially with drivers aged over 50. Concluding from this study the information for the driver must be presented in an appropriate way, e.g. by using audio messages. When information is presented on the screen, it has to be as clear and simple as possible, in order to minimise the number of distractions and the amount of time needed for understanding its meaning [26].

In a stationary vehicle the system needs to meet other requirements. Starting and stopping the services, as well as changing the preferences has to be simple and easy to learn. As many users might be familiar with Global Positioning System (GPS) navigation systems, application design and usage should be oriented at these systems to strive for consistency. Consistency is one of the "8 Golden Rules" proposed by Shneiderman et al. to increase the learnability and fun when interacting with a User Interface (UI) [27]. Consistency guarantees, that styleguides and conventions already applied for navigation systems are reused for *DriveAssist*.

Increasing usability means to use adequately big button sizes and a flat menu hierarchy because the application might be used in vibrating or poorly illuminated environments (vehicles). The interface should also be graphically appealing to the user and logically structured in order to allow an easy orientation within the application at all times. This is necessary because users might need to change system settings in time critical situations, e.g. when stopping at traffic lights.

5.1.2. Functional requirements

On the functional level, some other requirements are defined. The application has to be able to use two information sources: information originating from Vehicle-to-X (V2X)-Communication (see Chapter 2) and information from a Central Traffic Service (CTS) (see Chapter 3). Designed to demonstrate the potential of V2X communication for increasing traffic safety and driving convenience, efficient methods need to be developed in order to present information to the user in an appropriate way. As V2X communication does not yet exist commercially, real information from central traffic services should be included to enrich the functional range and to analyse whether or not different sources can be merged in a useful way.

An Android device can host many different applications and might also be used for other purposes while the car is moving (e.g. by the co-driver). Traffic events occurring in the user's direct surrounding are given the highest priority because traffic safety can be seriously put at risk. Therefore, incoming messages should be analysed in the background all the time to be able to warn the user if necessary. Practically, the application has to function as receiver and decoder for Cooperative Awareness Message (CAM)s (see Section 2.1.3) and Decentralized Environmental Notification Message (DENM)s (see Section 2.1.3) over WiFi in the background. Concurrently, the current user location has to be obtained to evaluate whether a warning is relevant for the user or not.

To support CTSs, the application should request Hypertext Transfer Protocol (HTTP) servers and download, decode and process the received information. The system should be easily expandable to be able to include other similar services in the future.

Finally, a database should be set up for storing incoming messages. Received messages are analysed in real time, but the database allows processing old messages received in the past or looking at what was received (merely for debugging).

5.2. Structure and Components

Based on the requirements presented in the previous section, a prototypic Android visual information system has been implemented. This section explains the application structure and components in detail, as well as the underlying concepts. Code snippets are given for clarifying implementation issues if appropriate. The whole source code is available at the Distributed Multimodal Information Processing Group (VMI)[1]. All snippets are Java code.

The application named *DriveAssist* consists of several Android components (see Chapter 4). For explaining the functionality of the system, it was decided to use a communication diagram to clarify the interaction between the components. Hereby, the function of a component can be seen in the framework of the whole application.

The communication diagram of core functionalities is shown in Figure 5.1. "Core" refers to the fact that all components necessary for assuming the compliance of requirements are shown, whereas extra features (e.g. for debugging or testing) are left out. The latter are briefly explained in Section 5.2.11.

Figure 5.1 uses yellow boxes to symbolise Android Activities, orange boxes for Android Services and green boxes for helper classes. Red circles represent input/output interfaces.

The communication between the elements is shown by different line types. In order to keep it as simple as possible, all communication concerning starting and stopping are visualised with dotted lines. Whenever an element needs values that are customisable by the user it needs information from the *Preferences Activity*. All these paths are drawn with dashed lines. Other communication is shown with solid lines. When values are sent and received, they are embraced with square brackets, whereas commands are not embraced.

In the following, all components will be explained in detail.

5.2.1. *DriveAssist* Main Menu

The *Main Menu Activity* is shown in the middle of Figure 5.1. It implements the entry point of the application and the first Activity that comes up when starting it. A screenshot of the main menu is shown in Figure 5.2. Four big buttons dominate the view which make four central functionalities of the system available: starting and stopping background services (outer left), starting the map

[1]Fachgebiet Verteilte Multimodale Informationsverarbeitung, `http://www.vmi.ei.tum.de/`, last visited 28 February 2012.

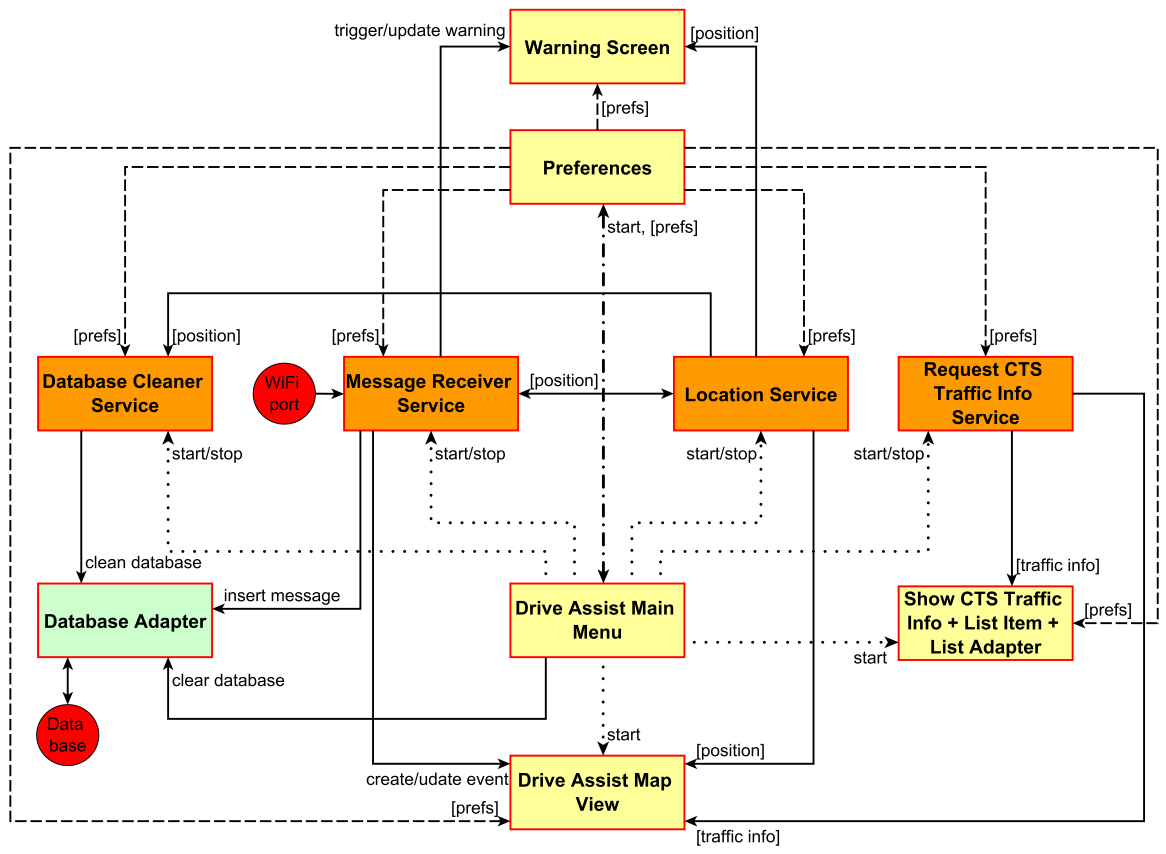


Figure 5.1.: *DriveAssist* Core Communication Diagram. Yellow boxes symbolise Activities, orange boxes represent Services and blue boxes stand for helper classes. Red circles are input/output interfaces. Dotted lines concern starting and stopping of components, dashed lines provide elements with information from preferences. Solid lines stand for all other communication.

view (inner left), showing traffic information from CTSs (inner right) and accessing the *Preferences Activity* (outer right). All these connections are symbolized with dotted arrows in Figure 5.1.

When pressing "Start Services" two basic services are always started automatically: the *Message Receiver Service* (Section 5.2.2) and the *Location Service* (Section 5.2.3). The *Database Cleaner Service* (Section 5.2.5) as well as the *Request CTS Traffic Info Service* (Section 5.2.4) are only started if they have been activated in the preferences. Therefore, the Activity needs information from the *Preferences Activity*. This way the user can use a less complex version without loosing the warning functionality of the whole system. The screenshot in Figure 5.2 shows the status when services have been started. The button has toggled to "Stop Services" and a notification is shown in the notification bar of the device (small car symbol in the upper left corner). The user is therefore aware of the system's status at all times, even when using other applications in the foreground.

When pressing the menu-button of the device, five menu options are displayed which are not represented in the communication diagram, except the "Clear database" function. They are subsidiary functions for testing and debugging the device and are briefly explained in Section 5.2.11. The option "Clear database" enables the user to clear the database manually. It is also possible to display an "About" dialog in order to access information about the development of the application and to contact the developer.

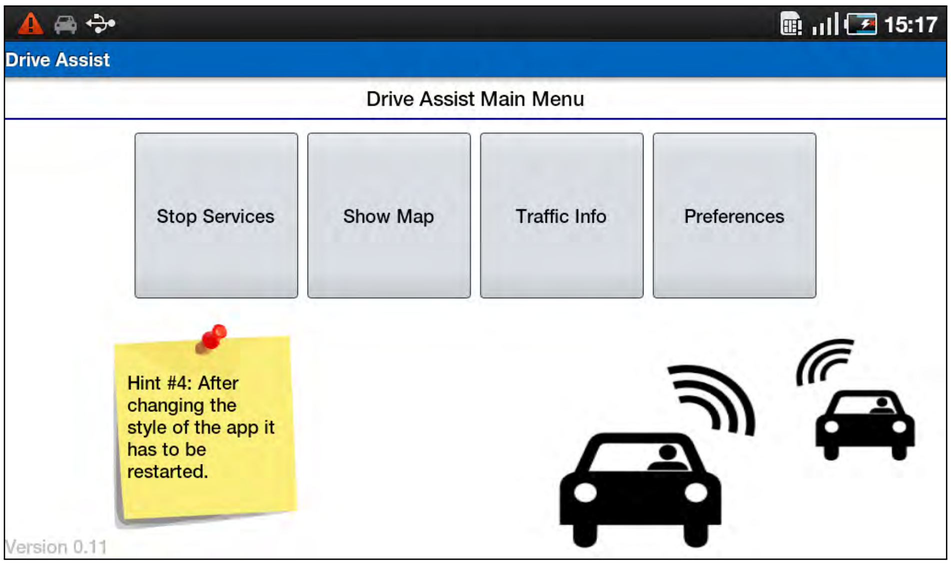


Figure 5.2.: *DriveAssist* main menu screenshot in landscape mode

If enabled in the settings, a yellow hint is displayed in the main menu containing short pieces of information of how to use the system while actually using it. They are randomly chosen from a selection at every start and can be blanked by tapping on it. The effect and usefulness of the hint is evaluated in the laboratory user test (see Chapter 7).

The portrait mode is also supported by the main menu.

5.2.2. Message Receiver Service

The *Message Receiver Service* is a central component of the application. It is an Android Service that runs in the background all the time, even if other Activities or other applications are in the foreground. This allows to analyse incoming messages and to warn the user when appropriate.

Receiving of Messages

The *Message Receiver Service* is responsible for receiving and processing V2X-messages. CTS messages are received by the *Request CTS Traffic Info Service* (Section 5.2.4). For receiving

V2X messages on a specified port using User Datagram Protocol (UDP) packets, a server thread `udpServerThreadMsg` implements a socket.

```
1  DatagramSocket socket = new DatagramSocket(Consts.UDP_PORT_MSG);
2
3  while (Thread.currentThread() == udpServerThreadMsg) {
4    byte[] buf = new byte[1024];
5
6    DatagramPacket packet = new DatagramPacket(buf, buf.length);
7    socket.receive(packet);
8
9    mHandler.sendMessage(Message.obtain(mHandler, 0, Byte_Tools.getSubByte(packet
         .getData(), 0, packet.getLength())));
10 }
11
12 socket.close();
```

Listing 5.1: Message receiver thread code snippet

Listing 5.1 shows the implementation of the message receiver thread. The UDP port can be set in `Consts.java` (line 1). While the thread is active, bytes from datagram packets are read from the socket and written to a buffer (line 6, 7). To decode the messages, a `Handler` object is used which processes every incoming message. Only the Protocol Data Unit (PDU) is forwarded using the function `getSubByte(byte[] array, int offset, int length)` from `Byte_Tools.java` (line 9).

Handling Incoming Messages

The `Handler` object is responsible for decoding the message and determining whether a CAM or a DENM has been received. Listing 5.2 shows a snippet of the handler's `handleMessage(Message msg)` method.

```
pbL1Container pbl1 = pbL1Container.parseFrom((byte[])msg.obj);
int msgType = pbl1.getMsgType();
  switch (msgType) {
    case Consts.CAM_MESSAGE: {
      pbCamPdu cam_rec = pbCamPdu.parseFrom(pbl1.getPayload(0).toByteArray());
      // [...]
      processCamMsg(cam_rec);
    } break;
    case Consts.DENM_MESSAGE: {
      pbDenmPdu denm_rec = pbDenmPdu.parseFrom(pbl1.getPayload(0).toByteArray()
          );
        // [...]
        processDenmMsg(denm_rec);
    } break;
```

Listing 5.2: Message handler code snippet

As a data serialisation format, Google's *Protocol Buffers* are used[2]. In line 1, the received messages are decoded to a `pbL1Container` object which is a customised container format containing (amongst others) the message type and the message PDU. The message type is determined (line 2) and CAMs and DENMs are parsed and processed separately. The message type integer codes are defined in `Consts.java`. *Protocol Buffers* original implementation is available under an open-source license. Libraries are available for several programming languages including Java.

Priority Mechanisms

Now that the CAM or DENM was successfully received, it can be processed. The next steps depend on whether the user has activated the map view or not. In case of an active map view, interesting events are forwarded to the *Map Activity* to be displayed. If the map view is inactive, the receiver service analyses whether a warning screen has to be displayed. The mechanisms for CAMs and DENMs are similar but differ in a few important details: warnings signalised by CAMs have a higher priority than warnings from DENMs. It is assumed that DENMs represent static traffic events (e.g. roadworks) and CAMs are emitted by moving events (e.g. an ambulance).

Thus, to decide which traffic event should be displayed in case of a warning screen (on the map all interesting events are displayed at the same time), three constraints considering priority have been made in the following order:

1. Moving events have a higher priority than static events because they can move towards the user and might need quick user interaction (e.g. driving to the side lane in case of an approaching ambulance). The danger of static events only depends on the user's movement. Hence, CAM-announced emergency vehicles have a higher priority then DENM-announced events and can even mask the latter.

2. Events towards which the distance to the user decreases have a higher priority than events which depart from the user because only approaching events will become relevant in the near future.

3. Near events have a higher priority than events which are further away, i.e. the closest event is the event to be displayed. Only the linear distance is considered, the route context is not taken into account.

[2]protobuf - Protocol Buffers, `http://code.google.com/p/protobuf/`, last visited 28 February 2012.

Following these constraints, the implementation needs a boolean variable set to true if a high priority event is displayed. In this case, DENMs are not evaluated until the high priority event has passed the user.

For explaining the mechanisms of how incoming messages are processed, the case of an incoming CAM will be presented. In case of an incoming DENM, the same mechanisms are implemented, but are not covered in detail in this thesis.

EGO CAM User Position

To determine the user position, the application can evaluate so-called EGO CAMs. These messages are emitted by the user vehicle and therefore have a known station ID. How the position for generating EGO CAMs will be determined is not yet fully standardised, but it is probable that the vehicle GPS receiver will be used. In our simulated environment, a CAM with a specified station ID can be used to direct the user vehicle (in our case, the station ID $= 0$ was chosen). The receiver service has to filter EGO CAMs from all other CAMs in order to extract the current position. It then informs the *Location Service* about position updates. The *Location Service* afterwards broadcasts the new position to the application. This step is done before continuing with the CAM analysis, i.e. EGO CAMs are not further analysed. In Section 5.2.3, details concerning location determination are given.

Warning Screen Algorithm

At first, it is explained how the *Message Receiver Service* processes CAMs in case of displaying a warning screen. All Non-EGO CAMs (messages from other traffic participants) are processed as shown in the flow diagram in Figure 5.3.

The algorithm is used for every incoming message at the entry point (START). At first, the position and identifier of the vehicle are extracted. Since the service knows the current user position, it can calculate the distance to the event. The current version only supports emergency vehicles. All other vehicle types are not considered. Whenever the algorithm reaches an exit point (END), the received message will be dismissed (unless it is not stored in the database).

Emergency vehicles are checked against a freeze list. This list contains identifiers of stations who have passed the user recently and hence should not be considered for the moment. The freeze list is cleared regularly by a timer (every 5 minutes). This allows reconsideration of these vehicles in the future.

The next step is to check whether the emergency vehicle is close enough to be relevant for the user. The user can customise this threshold to define at which distance the system shall warn the user about an approaching emergency vehicle. All vehicles outside this radius are not processed.

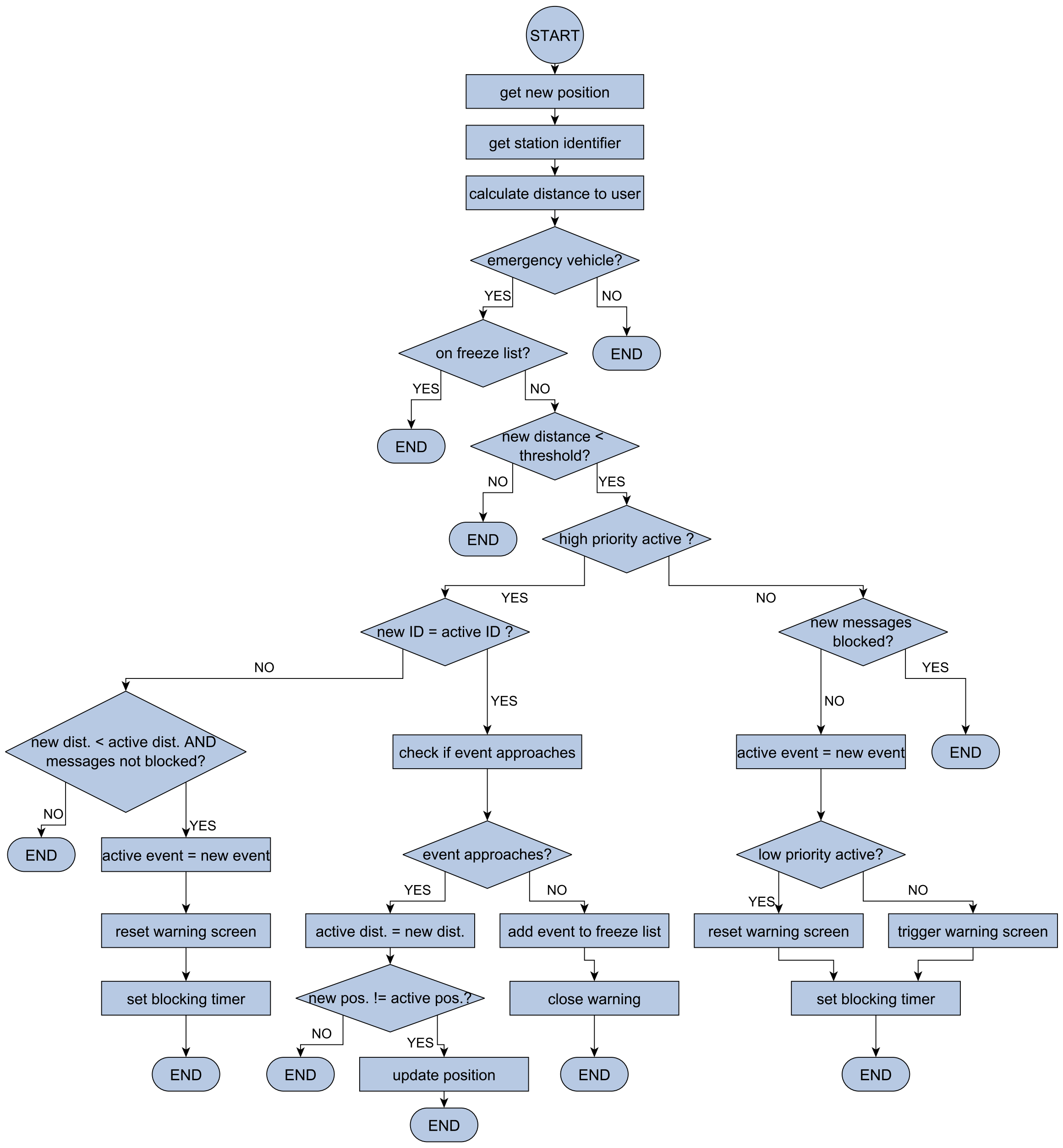


Figure 5.3.: Flow diagram of warning screen algorithm

From now on, it is important whether there is already an active warning screen or not. As explained above, emergency vehicles have a higher priority than static events. Therefore, it is checked if a high priority event (i.e. an emergency vehicle) is displayed at the moment. If this is not the case (as it would be when the application has started), it is checked whether new messages are allowed to be displayed or not. New messages are blocked for 5 seconds when a warning screen has been started recently to avoid sudden changes if many events occur at the same time. A timer is used to toggle the variable `newMsgBlocked` to false whenever it expires. The last branch refers to whether a low priority warning is active (from a static event) or not. In both cases, the emergency vehicle will be displayed due to its higher priority. But when a low priority warning is active, the same warning is replaced by the new event. The *Warning Screen Activity* is simply triggered when no other warning is active. In both cases, the blocking timer is set for five seconds.

If a high priority event is currently displayed, the received message could have originated from this event (new station identifier equals active station identifier[3]) or from another emergency vehicle. If it was sent from the event which is currently displayed, an algorithm is used to determine whether the event is still approaching. When it is still approaching, it is still relevant for the user and is updated with a new position (if the position has changed). Hence, the new position becomes active. If the active event does no longer approach, it has passed the user (from behind or from ahead). In this case, the warning screen can be closed. At the same time, the vehicle's identifier is added to the freeze list.

The last possible case is that the new event is not displayed in this instant. In this case, it is checked if it is located closer to the user than the active event and new messages are not blocked at the moment. If it is closer, it should be displayed and therefore the warning Activity is re-initialised with the new event.

This algorithm tries to model many possible traffic situations. In reality, diverse situations can occur and it is a very complex task to model all of them with a single algorithm. One key feature is the determination whether an event (static or movable) is approaching or not. Comparing subsequent distances is not sufficient as a single constraint because an event can still approach on a global scale even when the distance becomes greater for some time before decreasing again. Such a scenario is exemplarily shown in Figure 5.4. The change between the status "event approaches" and "event departs" is therefore limited to a small circle of $25\,\mathrm{m}$ around the user. If the event never touches this circle the warning will automatically be closed when the distance exceeds the predefined threshold.

At the same time, position updates cannot be assumed to be exact at every point of time. To absorb these inaccuracies, three subsequent distances are considered as shown in the code snippet in Listing 5.3.

[3]The identifier is the sequence number concatenated to the station ID.

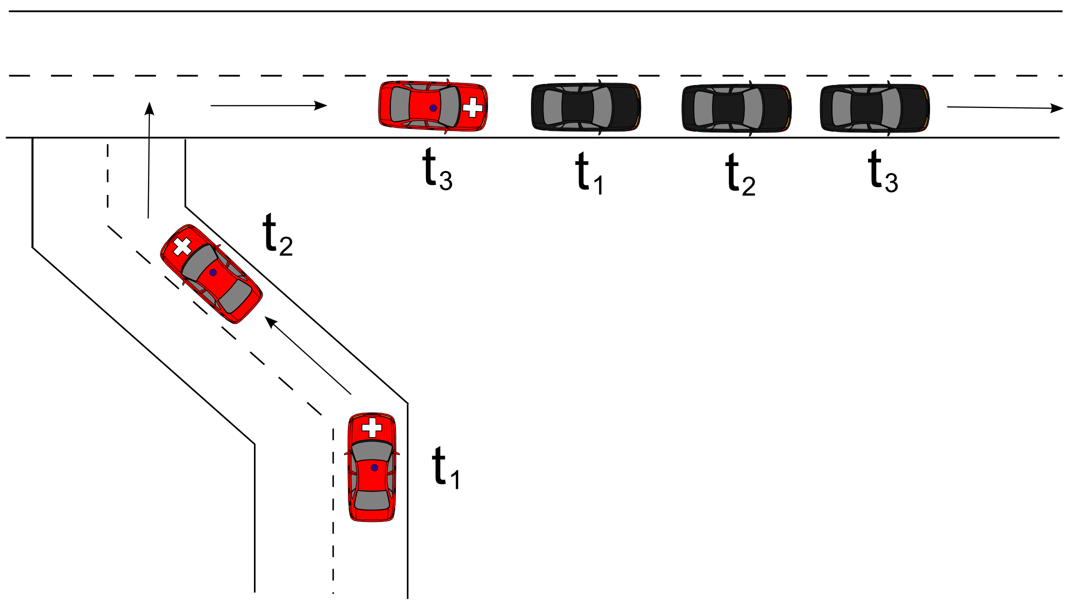


Figure 5.4.: Traffic scenario demonstrating an unsteady approaching of an emergency vehicle (red). The vehicle approaches steadily (t_1) with a greater speed than the user (black). Due to the course of the road, their distance increases temporarily (t_2) and decreases again (t_3).

```java
private boolean isEventApproachingCam(double _newUserEventDistance, double
    _activeUserEventDistance, double _beforeActiveUserEventDistance) {
    if (_beforeActiveUserEventDistance < 25.0 && _activeUserEventDistance <
        25.0 && _newUserEventDistance < 25.0) {
        if (_beforeActiveUserEventDistance < _activeUserEventDistance &&
            _activeUserEventDistance < _newUserEventDistance) {
            return false;
        }
    }
    return true;
}
```

Listing 5.3: Function to determine whether an event approaches or not

Two constraints have to be satisfied when an event is considered as departing: All distances have to be smaller than $25\,\mathrm{m}$. If this is the case, the newest distance (`_newUserEventDistance`) has to be greater than the active distance (`_activeUserEventDistance`) and the active distance has to be greater than the distance before the active distance (`_beforeActiveUserEventDistance`) simultaneously. Otherwise the function returns true which means that the vehicle is still approaching.

For static events (DENMs), a similar algorithm is used, which additionally includes, whether a high priority warning is active or not. Position updates for static events are also supported, even if they are very unlikely.

Map View Algorithm

If the user has activated the map view, the receiver service forwards all relevant events (both for CAMs and DENMs) to the *Map View Activity*.

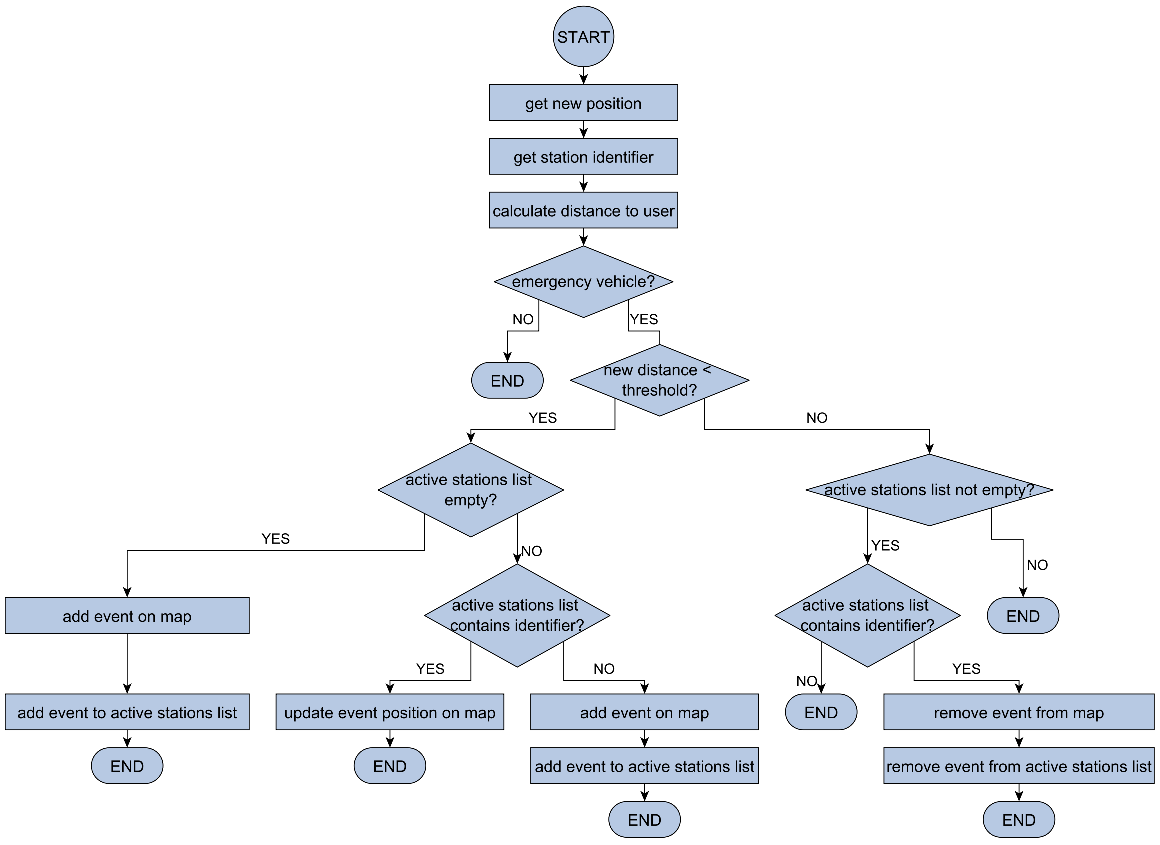

START
get new position
get station identifier
calculate distance to user
emergency vehicle?
NO
YES
END
new distance < threshold?
YES
NO
active stations list empty?
active stations list not empty?
YES
NO
YES
NO
add event on map
active stations list contains identifier?
active stations list contains identifier?
END
YES
NO
NO
YES
add event to active stations list
update event position on map
add event on map
END
remove event from map
END
END
add event to active stations list
remove event from active stations list
END
END

Figure 5.5.: Flow diagram of map view algorithm

A flow diagram of the implemented algorithm is shown in Figure 5.5. The beginning resembles the algorithm for the warning screen in Figure 5.3. Messages from emergency vehicles are analysed, if their distance to the user is smaller than a customisable threshold. The threshold for the map view can be seen as the radius of a circle around the user in which traffic events are displayed on the map ("circle of interest"). The threshold is not the same as used for the warning screen. Usually, it is larger ($500\,\mathrm{m}$ to $1000\,\mathrm{m}$) to overview the surroundings.

In case of received messages coming from an emergency vehicle which has entered the relevant radius, the algorithms checks the identifier of the new message against a list of active stations. This list contains all events (from CAMs and DENMs) which are currently displayed on the map.

When the list is empty (as it is when the application has started), the new event can be displayed on the map and added to the list. If the list already contains active stations (i.e. events) it is possible that the new message comes from an active station (in this case, the list contains its identifier). Its position on the map is updated. If the new message comes from a new station, it is added to the map and list as if the list was empty.

To guarantee that emergency vehicles or static events are removed from the map when they are no longer located in the "circle of interest", every incoming message that is outside the circle is also checked against the list of active stations (right part of the diagram). If the list is not empty and contains the identifier from the vehicle or event received (because it was displayed before), it is removed from the map and the list. The next message from the same station will not be considered because it is no longer in the list of active stations. When it enters the "circle of interest" a second time, it is displayed again.

This approach has two advantages: Firstly, it corresponds to the situation in reality. When driving on a motorway or country road, it is probable that other traffic participants who have passed the user will enter the "circle of interest" again. In this case, a list of permanently blocked stations would not be appropriate because stations entering and leaving the circle in short time frames would not be displayed but might be relevant. Secondly, this approach keeps the processing effort low. As only events in the "circle of interest" are displayed on the map, all other events do not need to be drawn. As especially drawing operations can cause high processing needs, this approach keeps the map reactive (also see Section 6.3.2).

If the user scrolls the map or zooms out, events outside the "circle of interest" will not be visible. For driving activities this disadvantage is tolerable.

A table of incoming and outgoing Intents of the Service can be found in Appendix A.

Calculating the Distance

The system has to perform distance calculations between the user and traffic events. The position of both the user and the event are given in geographic coordinates (latitude and longitude) and the calculation has to be done on a sphere. In order to calculate the shortest distance between two points on the surface of a sphere the great-circle distance is used. To calculate this distance, Sinnott proposed the Haversine formula [28] which uses the law of haversines:

$$\text{haversin}\left(\frac{d}{r}\right) = \text{haversin}(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)\,\text{haversin}(\psi_2 - \psi_1) \tag{5.1}$$

The Versine or versed sine of an angle α is $1 - \cos(\alpha)$. The Haversine is half of the versine, i.e.

$$\text{haversin}(\alpha) = \frac{1-\cos(\alpha)}{2} = \sin\left(\frac{\alpha}{2}\right)^2 \tag{5.2}$$

To calculate the distance d between two points on a sphere with a radius r, one can solve Equation 5.1 to d by using Equation 5.2. This leads to the equation

$$d = 2r\arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2-\phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\psi_2-\psi_1}{2}\right)}\right) \tag{5.3}$$

where ϕ_1 and ϕ_2 are the latitudes of point 1 and point 2, and ψ_1 and ψ_2 are the longitudes of point 1 and point 2. For r the earth radius of $6371\,\text{km}$ is used.

Listing 5.4 shows the implementation of the formula as it is used in the application.

```java
public static double calcDistance(float _userPosLat, float _userPosLong, float
    _eventLat, float _eventLong) {
    float R = 6371; // earth radius
      double dLat =  Math.toRadians(_eventLat - _userPosLat);
      double dLon =  Math.toRadians(_eventLong - _userPosLong);
      double a = Math.sin(dLat/2) * Math.sin(dLat/2) + Math.cos(Math.toRadians(
          _userPosLat)) * Math.cos(Math.toRadians(_eventLat)) * Math.sin(dLon/2)
           * Math.sin(dLon/2);
      double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    double dist_km = R * c;
    return dist_km * 1000;
}
```

Listing 5.4: Distance calculation using the Haversine formula

The formula calculates precise results. The distance is returned in meters.

5.2.3. Location Service

The other central Service besides the *Message Receiver Service* is the *Location Service*. In order to relate incoming messages to users, their position must be known by the system at every time. The *Location Service* is responsible for determining the user position with one of three supported determination methods in the background. When the user presses "Start Services", the *Location Service* starts and begins to broadcast the position via an Intent towards the application.

The determination method can be chosen by the user in the Preferences (Section 5.2.10). The possible determination methods are:

- **Device GPS:** The device GPS location provider is used. A position update will be broadcast whenever available, but at the earliest every $10\,\mathrm{m}$ or $60\,\mathrm{s}$ (can be customised in `Consts.java`).
- **Network provider:** The network location provider is used. The position is determined using registered WiFi access points or cell tower triangulation. With this option, it is possible to localise the user when no GPS signal is available, e.g. indoors. A position update will be broadcast whenever it is available, but at the earliest every $10\,\mathrm{m}$ or $60\,\mathrm{s}$ (can be customised in `Consts.java`).
- **EGO CAM:** The *Message Receiver Service* is used to obtain the user location (see Section 5.2.2, "EGO CAM User Position"). In this case, the *Location Service* does not actively determine the position. Location updates are received by the Location Service and forwarded towards the application. This concept was chosen to keep the location responsibility at the Location Services' site.

It is also possible to set a static user position in the *Preferences Activity*. The latitude and longitude can be chosen and are broadcast regularly using a timer task. Static positions are merely used for debugging.

A table of incoming and outgoing Intents of the *Location Service* can be found in Appendix A.

5.2.4. Request CTS Traffic Info Service

DriveAssist is able to display traffic information from CTSs (see Chapter 3). The *Request CTS Traffic Info Service* is responsible for requesting this information regularly from a server and for broadcasting it to Activities which have registered a Broadcast Receiver. As there are many different traffic information sources available, the Service can be customized to request the desired source.

A server proxy requests the data from the provider and makes it available for the application. Via an HTTP request a JavaScript Object Notation (JSON) object can be obtained, including all current traffic messages available in form of JSON nodes. JSON was designed to represent simple data formats in a serial manner. It is language-independent (despite its relation to JavaScript), open-source and can be seen as an alternative to Extensible Markup Language (XML).

The Service requests a new JSON object when started, when requested by another component explicitly by an Intent or regularly in a customisable interval using a timer (see Section 5.2.10). The server proxy address is specified in `Consts.java`. A string is received after an HTTP request which is then converted to a JSON object. Different error types, such as HTTP timeouts, parsing or conversion errors, are handled by the Service by reporting them towards the system. Thus, they can be presented to the user directly in the currently visible Activity.

A table of incoming and outgoing Intents of the *Request CTS Traffic Info Service* can be found in Appendix A.

5.2.5. Database Cleaner Service

To complete the explications about the applications' Services, this section focuses on the *Database Cleaner Service*. If enabled in the Preferences, it starts when the user presses "Start Services".

With the current user location (received by a Broadcast Receiver) the *Cleaner Service* accesses the database via the *Database Adapter* (Section 5.2.9) to delete all entries which are outside the circle defined by the cleaning radius. That keeps the database small in size and prevents its overfilling. The interval between two deleting actions can be customised in the Preferences (from $500\,\mathrm{ms}$ to $30.000\,\mathrm{ms}$), as well as the radius (from $500\,\mathrm{m}$ to $3000\,\mathrm{m}$).

To calculate the distance to all database entries, the imprecise method of using the Pythagorean Theorem with a fudge factor was chosen. The fudge factor roughly compensates projection errors because the distances are calculated on a sphere. The loss of precision can be accepted because the precise distance is only needed in the direct neighbourhood of the user[4]. It is outweighed by the fact that this method can directly be processed by the SQLite database because it can be formulated as a SQL statement. SQLite does not support trigonometric functions. Thus, it does not allow using more complex functions, such as the Haversine formula. The fudge factor $f = \cos(\phi)^2$ takes into account how distant the point with latitude ϕ is away from the equator. For further details it is referred to the source code.

The *Database Cleaner Service* has a lower priority than every other Service or Activity that accesses the database. That means if the Service tries to access the database to delete some messages while another thread has blocked it, the deleting task will simply be left out once.

A table of incoming and outgoing Intents of the *Database Cleaner Service* can be found in Appendix A.

5.2.6. Warning Screen

The *Warning Screen Activity* cannot be started explicitly by the user. It is a slave element of the *Message Receiver Service* and fully controlled by it. Moreover, it is a passive element which does not require user interaction. Its only purpose is to display a warning to the user. A screenshot of the *Warning Screen Activity* is shown in Figure 5.6.

[4] For distances less than $20\,\mathrm{km}$ the use of the Pythagorean Theorem will result in an error of less than $20\,\mathrm{m}$ for latitudes less than $50°$.

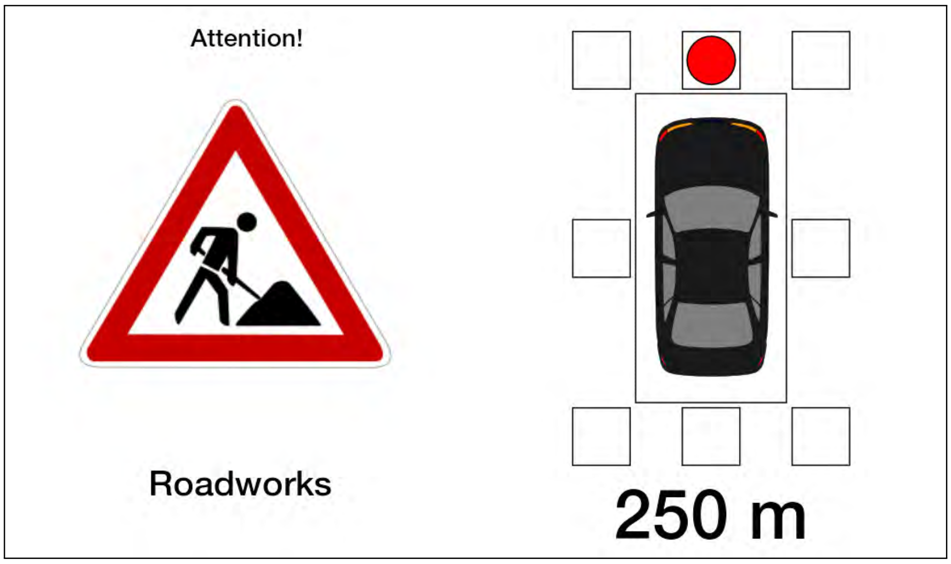


Figure 5.6.: *DriveAssist Warning Screen Activity* screenshot in landscape mode

The warning shows a construction site which lies $250\,\mathrm{m}$ ahead of the user. The direction in which to expect the event is calculated relatively to the user and indicated by a red dot. It can take one of eight possible positions around the car, representing angles of $45°$.

Initialisation of the Warning Screen

When the warning screen is started, it is initialised with a background and cause string (located at the bottom left side of the screen) corresponding to the type of the event. All Day-1 Use Cases as explained in Section 2.2.2 are supported with a different symbol. All images of the basic set are shown in Figure 5.7. Common standardised traffic signs were used preferably[5] (Image 1, 3 and 6). In case there was no official sign available, appropriate sign were designed (Images 2, 4 and 5). These creations orient on the typical pictograms using common elements, such as the car symbol in image 5. This helps the user to rapidly understand the meaning of unfamiliar signs. The red triangle symbolising "Attention" was used in all cases because users are familiar with its appearance.

At the same time, the distance to the event is set. The distance is calculated by the *Message Receiver Service* and is added to the Intent which starts the *Warning Screen Activity*, as well as the integer code for the cause and other parameters.

A Text-to-Speech (TTS) engine is initialised (corresponding to the devices' language) and an acoustic warning is spoken respecting the cause and including the distance. An example sentence

[5]Source: Wikimedia Commons, `http://de.wikipedia.org/wiki/Bildtafel_der_Verkehrszeichen_in_Deutschland`, last visited 28 February 2012.

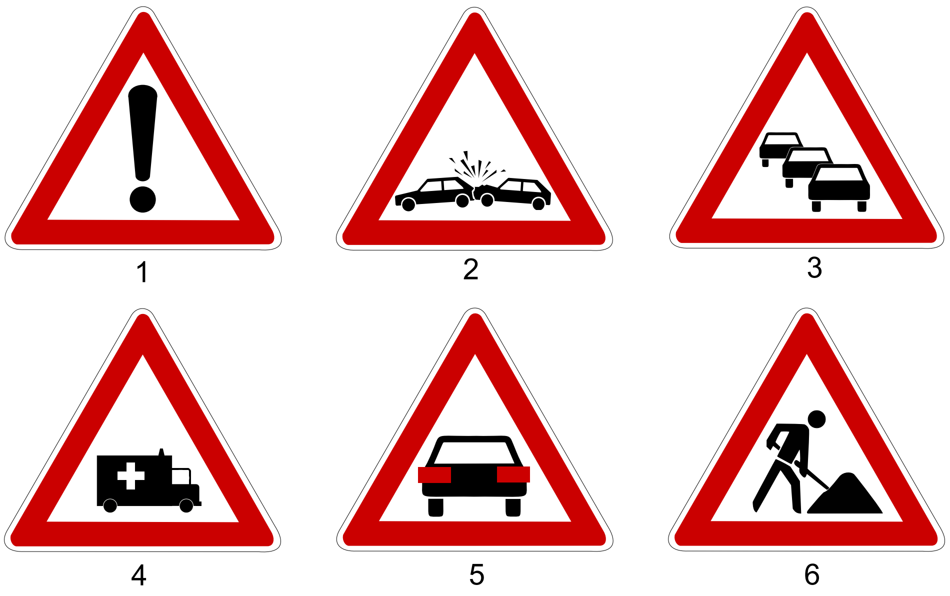


Figure 5.7.: Set of traffic signs for V2X-alerted traffic events. The symbols are used to inform about a hazardous location (1), a stationary vehicle (2), a traffic congestion (3), an emergency vehicle (4), a sharp braking vehicle (EEBL, 5), and roadworks (6). The images 1, 3 and 6 are standardised German traffic signs, images 2, 4 and 5 were specifically designed.

corresponding to Figure 5.6 would be "Attention! Roadworks in $250\,\mathrm{m}$!".

Closing the Warning Screen

As long as the *Warning Screen Activity* is active, the distance to the event and the direction dot are updated within the Activity. When the user has passed the event, the *Warning Screen Activity* is closed explicitly by the *Message Receiver Service* (the master) by sending an Intent. When the *Warning Screen Activity* receives this Intent, it announces its closure by another Intent and calls its function `finish()`, which closes the Activity. The announcement (warning was closed) is in turn needed by the *Message Receiver Service* in order to know whether a warning is active or not.

If this mechanisms fails (e.g. because the event is never set to departing), the *Warning Screen Activity* implements a security barrier: It will automatically close if the displayed event is no longer in the "circle of interest". It announces its closure in this case, too.

Updating the Distance

The distance is updated whenever a new user location update is received. The distance is calculated using the Haversine Formula introduced in Section 5.2.2. It is rounded to multiples of

$50\,\mathrm{m}$. If it has changed compared to the old rounded distance, the new distance is set. Once in a lifecycle of a *Warning Screen Activity* another acoustic warning saying e.g. “Roadworks are nearby!” is evoked when user and event are $50\,\mathrm{m}$ close for the first time. The repetition of the acoustic warning shall focus the user's attention to the event once more because it is in direct proximity.

Updating the Direction

Similar to the distance, the direction indication is also updated on every user position update. The direction is represented by an angle α between the user and the event, as shown in Figure 5.8.

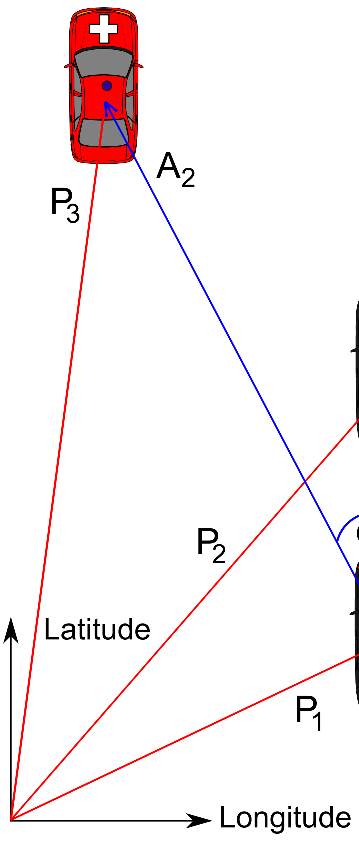


Figure 5.8.: Relative direction calculation for warning screen Activity

With the assumption that the vector in driving direction A_1 can be calculated when taking the position of the car at two subsequent location update times t_0 and t_1, it is possible to calculate α when constructing the second vector A_2 from the user position at t_0 and the current event position (blue vectors). All positions are given in the World Geodetic System 1984 (WGS84) reference system (with latitudes and longitudes, red vectors P_1 to P_3) which are needed to construct A_1 and A_2. As two subsequent location updates are needed, both have to be updated permanently.

The scalar product is used to calculate the angle. Afterwards, the vector product is used to check if the angle was calculated in the mathematically positive sense (return angle) or negative sense (return $360°$ - angle). This ensures that the angle is related from $0°$ to $360°$ for eliminating ambiguity.

The calculated angle is rounded to a multiple of $45°$ and the appropriate image with a corresponding red dot is set. When event and user are more than $50\,\mathrm{m}$ close to one another, one cannot rely on the precision of the positions. It is not yet possible to determine whether the event passes e.g. by the left or the right of the user. Hence, it was chosen to use an additional direction indication called "central" when the event is very close. No red dot is shown, but a red box around the car. This, together with a TTS warning, should increase the driver's attention when being informed that the event is very close.

When observing the red dot in a simulated scenario, one can observe that it sometimes tends to "bounce" from one position to another and back. This can occur when the angle lies at a rounding border and the event position and user position are updated with different frequencies. Whether or not this effect is disturbing for the user will be evaluated in the laboratory user test in Chapter 7.

A table of incoming and outgoing Intents of the *Warning Screen Activity* can be found in Appendix A.

5.2.7. *DriveAssist* Map View

The *Map View Activity* represents an active element of the system which can explicitly be started by the user in the main menu. The cartographic material used by the application comes from OpenStreetMap (OSM). The open-source mapping tool *mapsforge* is used for rendering the material and creating a map view. With the help of a small library and compact map data files stored on the devices' Secure Digital (SD) card, it offers fast on-device rendering of OSM data and the user is not dependent on a working Internet connection. Multi-touch gestures, such as pinch-to-zoom, are supported. Dynamic rotation of the map is not yet possible in the current version 0.2.4. The user can choose a supported German Federal State in the Preferences which can then be displayed in the *Map View Activity*. New areas can be downloaded and the maps can be updated via the project homepage[6].

Displaying the User Position

As shown in Figure 5.9, the current user position is marked with a small car symbol. Users are familiar with this symbol as similar symbols are used in conventional navigation systems. The

[6] mapsforge, `http://code.google.com/p/mapsforge/`, last visited 3 March 2012.

position is updated on every position update from the *Location Service* and the symbol is redrawn. Furthermore, the map is centred to the new position in order to keep it clearly visible all the time.

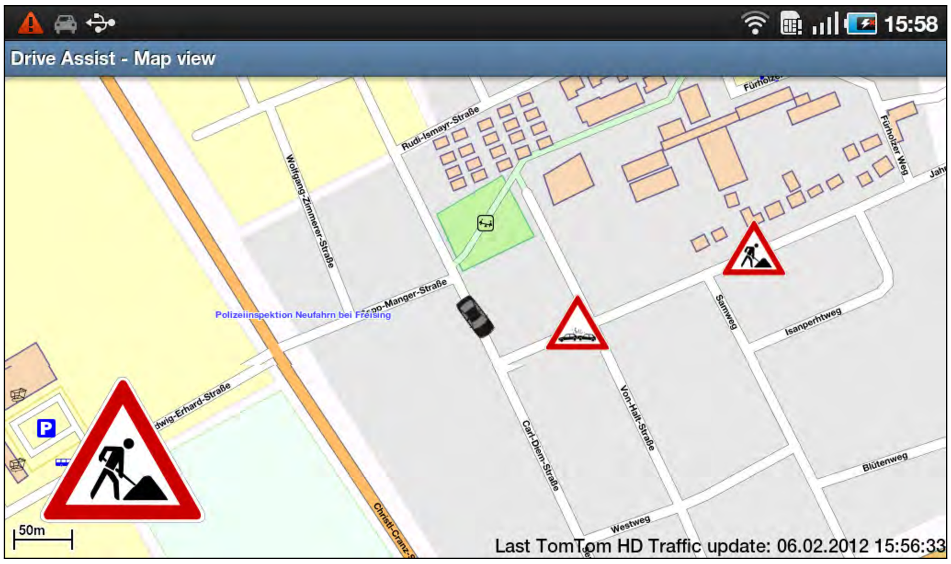


Figure 5.9.: *DriveAssist* map view screenshot in landscape mode

Also observable in Figure 5.9 is the fact that the map remains static while the car is rotated. This is unusual compared to modern navigation systems and more difficult to interpret. As mentioned above, the current mapsforge version does not support map rotation. It would be possible to turn the whole View holding the map, but in this case all texts and numbers on the map would turn, too. Thus it was decided to turn the car and test the impact on the user in the laboratory user test (Chapter 7). As soon as it is possible to rotate the map, it can be integrated as the necessary calculations are already done for rotating the car.

For rotating the car symbol, two components are needed: A rotation angle and afterwards a rotated version of the bitmap image. To calculate the rotation angle, the same function introduced in Section 5.2.6 will be used. The difference is that the vector from the user to the event is substituted by a vector from the user position at t_0 to north direction. This vector is the reference to the map which is also oriented north. When again taking two user positions at two subsequent times (t_0 and t_1), the latitude of the user position at t_0 is simply extended by $0°1'23''$ ($= 0.0004$ rad) to get a vector showing in the north direction. The value can be converted to meter and equals approximately $44.4\,\text{m}$. This value has experimentally proven to guarantee robust calculations. The resulting small projection error can be neglected.

With the rotation angle, it is possible to rotate the basic bitmap of the car before it is drawn. As bitmaps need a huge amount of memory, a mechanisms was implemented to save as much memory as possible. The implementation is shown in Listing 5.5.

```
1 private Drawable getRotatedDrawable(float rotationAngle, int requiredSize) {
2     Bitmap bmpOriginal = decodeFile(getResources().openRawResource(R.drawable.
          ego_icon_car), requiredSize);
3     Bitmap bmResult = Bitmap.createBitmap(bmpOriginal.getWidth(), bmpOriginal.
          getHeight(), Bitmap.Config.ARGB_4444);
4     Canvas tempCanvas = new Canvas(bmResult);
5     tempCanvas.rotate(rotationAngle, bmpOriginal.getWidth()/2, bmpOriginal.
          getHeight()/2);
6     tempCanvas.drawBitmap(bmpOriginal, 0, 0, null);
7     Drawable rotatedDrawable = new BitmapDrawable(bmResult);
8     return rotatedDrawable;
9 }
```

Listing 5.5: Rotation of the user position symbol code snippet

In line 2 , the input file containing the symbol is decoded. This is done in two steps: First of all, only the image size is determined, without decoding the whole image. Afterwards, the variable `requiredSize` is used to determine the scale of the image, which should be to the power of 2. When this is done, the image is directly decoded to the required size which saves memory.

Now, a new bitmap can be created to store the rotated version of the image on a Canvas (line 3 and 4). The Canvas is rotated (line 5) and drawn to a bitmap. The rotated version is returned and can be drawn on the map.

Displaying V2X Information

All V2X information relevant for the user are forwarded to be displayed on the map if the user has activated it. For that purpose, the map announces that it was started or stopped by an Intent which makes it possible to switch between the *Map View Activity* and the *Warning Screen Activity* at runtime. For traffic events alerted by V2X communication, a list of overlays named `mapOverlays`, is managed by the Activity containing all events displayed at that moment. Adding events to and removing events from the list is done by a Broadcast Receiver which listens for corresponding Intents from the *Message Receiver Service* . Whenever the set of events changes (position update of an event, addition/removal of an event, etc.), the map is redrawn. It is also checked if an expiry time is set for an event and if this is the case, expired events are removed from the map.

When a new traffic event enters the user's "circle of interest" around the user, a TTS engine is used to inform the user that a new traffic event has been detected and is now displayed on the map. Additionally, the appropriate traffic sign from Figure 5.7 is displayed for $8\,\mathrm{s}$ (customisable in `Consts.java`) in the bottom left corner of the screen. Hence, the user knows what type of event is active in his /her surrounding and can check on the map whether it lies on his/her route since the same symbol is used in a smaller version directly at the event's location (see Figure 5.9). The

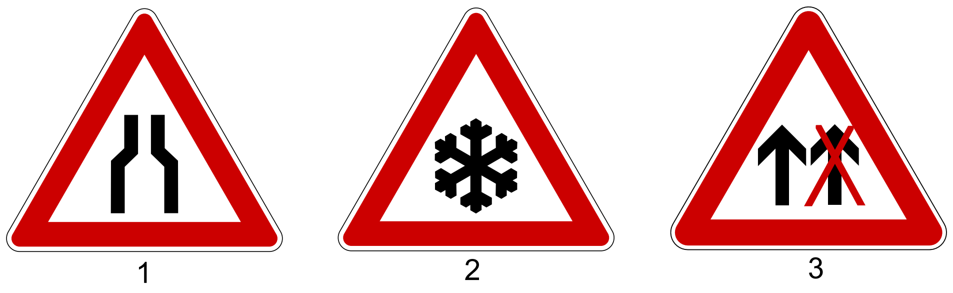


Figure 5.10.: Set of additional traffic signs for CTS-alerted traffic events. The symbols are used to report narrow lanes (1), icy roads (2) and blockings of right lanes (3) (left lane blockings respectively). The images 1 and 2 standardised German traffic signs, image 3 was specifically designed.

view on the map is compared to conventional navigation systems more zoomed out in order to give the user a broad overview about his/her surroundings. A new route can be planned immediately by the driver because all events around the current position (not only the events on the planned route) are visible.

When the user taps an event on the map, a Dialog is shown stating that the event was alerted by V2X communication. It is thinkable that additional information could be shown in this Dialog in the future, such as details to the event.

Displaying CTS Traffic Information

Information from CTSs can also be shown on the map if activated. The map requests a new JSON object from the *Request CTS Traffic Info Service* (see Section 5.2.4) when it is created and parses all events to `OverlayItem` objects which are shown on the map. In order to assign a symbol to every event, a classifier was developed. The classifier is based on phrase occurrences in the "description" field of every CTS message. As the vast majority of the messages reports traffic congestions, slow moving traffic, or similar events with repetitive phrases, the image 3 from Figure 5.7 is used very often. Key phrases for this symbol in CTS messages are "stationary traffic", "queuing traffic" and "slow traffic". For other phrases other symbols are used. Besides the basic set of traffic signs, additional signs are used for CTS traffic info[7]. The additional set of signs is shown in Figure 5.10.

All classified phrases can be found in the source code. When tapping an event, a Dialog is created which shows all information coming from the source. This can, for example, be the name of the street where the event occurs, a reference to where the event starts, or where it ends. The last update time of the information is shown in the bottom right corner of the map.

[7]Source of standardised German traffic signs: Wikimedia Commons, `http://de.wikipedia.org/wiki/Bildtafel_der_Verkehrszeichen_in_Deutschland`, last visited 28 February 2012.

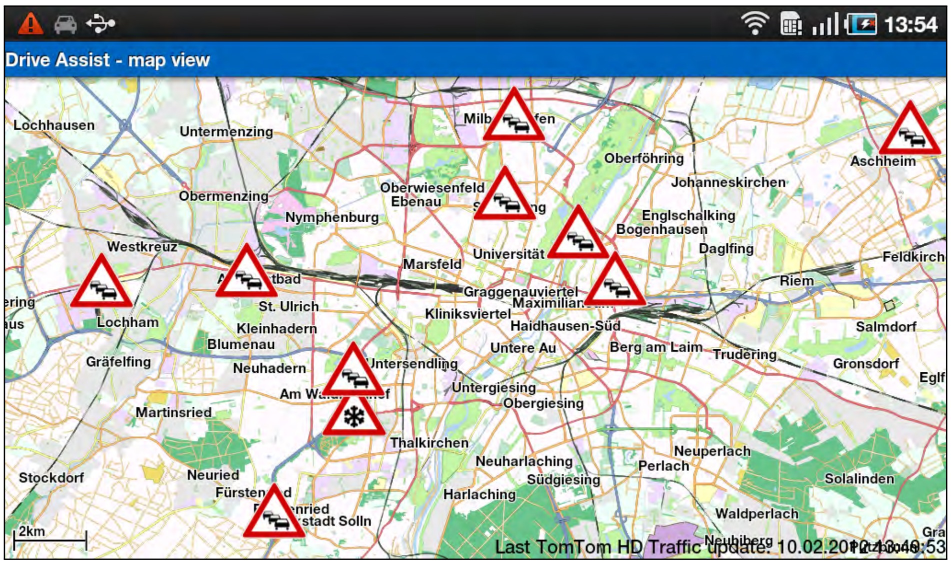


Figure 5.11.: *DriveAssist* map view screenshot in landscape mode showing traffic information from a central traffic service.

A screenshot of how CTS information is displayed on the map is shown in Figure 5.11. The user position can be displayed, too (ignored in the figure). The user has zoomed out to overview the whole Munich region. Whenever the *Request CTS Traffic Info Service* broadcasts a new JSON object, all events on the map are redrawn.

Map Overlays

Map overlays have been implemented for drawing symbols on the map. V2X-alerted events, CTS-alerted events, and the current user position can be displayed. For each of the three types, an own `ItemizedOverlay` class has been created, which is added to the map view during initialisation. For every item of CTS-alerted events and the user position, the native Android `OverlayItem` class is used. The overlay holds a list of overlay items and also implements the functions to add, remove, or update the items. Additionally, it handles the "onTap"-event (a Dialog is created) which is evoked when the user touches an item on the map.

For V2X-alerted events, the `OverlayItem` class was customised: An identifier string was added as a member which equals the one used in the *Message Receiver Service*. The overlay class implements functions to find an item based on its identifier and remove it from the map or update its position. Certain single items can be modified, which is not possible in both other overlays.

A table of incoming and outgoing Intents of the *Map View Activity* can be found in Appendix A.

5.2.8. Show CTS Traffic Info

Besides the *Map View Activity* and the *Preferences Activity*, the *Show CTS Traffic Info Activity* is the last Activity which can be started explicitly by the user from the main menu ("Traffic Info"-button). It is a list view Activity showing all received CTS traffic information while using one list item for every received message.

It requests a new JSON object from the *Request CTS Traffic Info Service* when starting, receives it and then parses it to a `CtsListItem`. The `CtsListItem` has been designed to be displayed in a list with the help of a list adapter class, implemented in `CtsInfoArrayAdapter.java` extending `ArrayAdapter<CtsListItem>`. Every list item was designed in XML and is inflated in the adapter class.

The list helps the user to check all reported traffic events before driving starts. An overview is given in a compact format to verify all information at the same time. When displayed on the map, the information is distributed and every single event needs to be touched in order to get additional information. This motivates both approaches at the same time and increases the usability and accessibility of CTSs.

The user can manually refresh the list by using the menu-button of the device. Otherwise, it is refreshed at every start or when a new update is broadcast automatically. The Activity needs information from the Preferences in order to change its style and to verify whether the *Request CTS Traffic Info Service* has been enabled.

A table of incoming and outgoing Intents of the *Show CTS Traffic Info Activity* can be found in the Appendix A.

5.2.9. Database Adapter

The database adapter is a helper class which implements all functions related to database access. It is responsible for creating the database tables and manages all requests from outside concerning reading, writing, and deleting entries in the SQLite database.

The database is designed to store CAMs and DENMs in two separate tables. The messages are not stored in total or as an object, but all required data to build a CAM/DENM is extracted and stored separately in a row. Thereby, information loss is encountered when saving a message in the current version because optional information is not stored. In return, the database is kept small in terms of column dimension and contains all information required in the first place.

The database can be activated and deactivated in the Preferences. In the available version, incoming V2X-messages are only stored by the system but never read for reasons of user warning (as all messages are analysed when received). The *Show V2X Messages Activity* (presented in

Section 5.2.11) requests the database in order to show all received messages during the process of development and debugging but not for direct user benefit. In future versions, the data can be used to "watch the past", for example in order to predict the route of a traffic participant and warn before a potential collision.

It is also possible to get a cursor on all messages with a certain feature from the Database Adapter, e.g. a special Station ID. The adapter offers the functions `getSelectedCamMsgCursor(String selection)` and `getSelectedDenmMsgCursor(String selection)` which return a cursor on all entries specified in the SQL statement `selection`. This can be used for filtering the database. The filter is used in the *Show V2X Messages Activity*.

Finally, the *Database Cleaner Service* uses the function `removeCamEntriesOutRange(float latitude, float longitude, double threshold)` (similar for DENMs) to delete all entries which are further away than a specified `threshold` around the user position. The user position is defined by a pair of `latitude`/`longitude` values. For details on how the distance is calculated, see Section 5.2.5).

5.2.10. Preferences

The application offers many customisations to fit the user's requirements. The central location to adjust the settings is the *Preference Activity*. Every system component which has customisable parameters can instantiate an `SharedPreferences` object in order to access the data. In Figure 5.1 all consumers are connected with a dashed line titled "[prefs]" to give an overview about what can be customised. The following list specifies what can be customised in each component:

***DriveAssist* Main Menu:**

- Services to be started: The *Database Cleaner Service* and the *Request CTS Traffic Info Service* can be enabled and disabled by the user.
- Application style: The color scheme of the Application (App) can be adjusted to the user's preference.

Message Receiver Service:

- Database support: Incoming messages are stored in the database. The database support can be disabled.
- Distance at which users are warned about a traffic event when using the warning screen.
- Radius around the user at which traffic events are shown on the map.

Location Service:

- Location determination method: The current user location can be obtained by using the device GPS, the network provider (or WiFi access points) or EGO CAMs (see Section 5.2.3).
- Fixed position: The user can enable a fixed position which will be broadcast regularly. In this case, the latitude and the longitude have to be specified.

Database Cleaner Service:

- Database cleaning interval .
- Database cleaning radius.

***DriveAssist* Map View:**

- Central Traffic Services support: CTSs can be enabled and disabled to be shown on the map.
- Map region to be shown.
- Application style: The color scheme of the App can be adjusted to the user's preference.

Warning Screen:

- Distance at which users are warned about a traffic event when using the warning screen.

Request CTS Traffic Info Service:

- Request interval: Interval at which the HTTP request is done.

Show CTS Traffic Info:

- Central Traffic Services support: CTSs can be enabled and disabled to be shown in the list.
- Application style: The color scheme of the App can be adjusted to the user's preference.

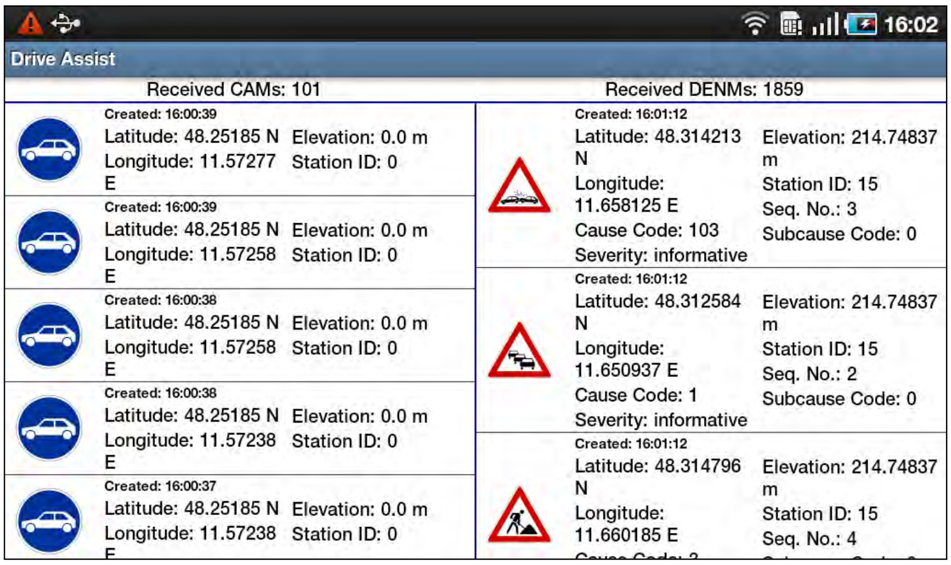
The *Preference Activity* accesses shared preferences to adjust the applied style of the application. The elevated number of customisable parameters enables users to personalise the application and to adjust it to their personal requirements. An example is the distance at which the system warns the user about a traffic event: People who want to be informed well in advance of the event, can increase the distance. People who want minimal distraction can decrease it.

5.2.11. Secondary Functionalities

As the application *DriveAssist* is a prototype and still under development, some secondary functionalities have been implemented in order to make testing and debugging more comfortable. These secondary elements do not contribute to better usability or a wider spectrum of functions for the user, but help the developer during the development process. All of them are callable by pressing the "menu"-button in the main menu. A communication diagram showing the secondary functionalities is provided in Appendix B.

Show V2X Messages

The *Show V2X Messages Activity* offers a list to show V2X-messages stored in the database. By setting filters via the menu, the displayed messages can be filtered to a certain type of cause or a certain station. In landscape mode, both message types are shown next to each other as depicted in Figure 5.12.


16:02

Drive Assist

Received CAMs: 101

Created: 16:00:39
Latitude: 48.25185 N Elevation: 0.0 m
Longitude: 11.57277 Station ID: 0
E

Created: 16:00:39
Latitude: 48.25185 N Elevation: 0.0 m
Longitude: 11.57258 Station ID: 0
E

Created: 16:00:38
Latitude: 48.25185 N Elevation: 0.0 m
Longitude: 11.57258 Station ID: 0
E

Created: 16:00:38
Latitude: 48.25185 N Elevation: 0.0 m
Longitude: 11.57238 Station ID: 0
E

Created: 16:00:37
Latitude: 48.25185 N Elevation: 0.0 m
Longitude: 11.57238 Station ID: 0

Received DENMs: 1859

Created: 16:01:12
Latitude: 48.314213 N Elevation: 214.74837 m
Longitude: 11.658125 E Station ID: 15
Seq. No.: 3
Cause Code: 103 Subcause Code: 0
Severity: informative

Created: 16:01:12
Latitude: 48.312584 N Elevation: 214.74837 m
Longitude: 11.650937 E Station ID: 15
Seq. No.: 2
Cause Code: 1 Subcause Code: 0
Severity: informative

Created: 16:01:12
Latitude: 48.314796 N Elevation: 214.74837 m
Longitude: 11.660185 E Station ID: 15
Seq. No.: 4

Figure 5.12.: *DriveAssist* Screenshot of the Show V2X Messages Activity. Stored CAMs are shown on the left, DENMs on the right in landscape mode.

In portrait mode, only one message type is displayed at once, but can be toggled by tapping the label above the list.

For displaying messages stored in the database, the *Database Adapter* (Section 5.2.9) is used, as well as two list item classes (`CamListItem` and `DenmListItem`) and two list adapters (`CamMessageArrayAdapter` and `DenmMessageArrayAdapter`). The implementation works similar to the case explained for CTS messages in Section 5.2.8.

Create CAM/DENM

Two more Activities are available to manually create CAMs or DENMs. A simple mask is provided in which all required fields for a CAM or DENM need to be inserted. When pressing "Create CAM" or "Create DENM" the message is inserted into the database. The Activities have been designed to test database operations and message encoding. The latter can be used in future implementations for messages to be sent out by the device.

5.3. Software Modularity

The whole application is designed to satisfy the principles of software modularity. All Services and Activities can be replaced or changed without affecting the rest of the application. Only the Intents required for communicating data from one element to another need predefined interfaces. All Intent dependencies are provided in Appendix A.

XML-based components, such as strings (enable multiple language support), layouts and colours can be changed centrally as it is common for Android App development. As these files are independent from the rest of the application, they can be modified without taking care of software dependencies. All customisable constants of the App are configured in the `Consts.java` file.

For supporting other CTSs, the *Request CTS Traffic Info Service* (Section 5.2.4) needs to be modified (file `RequestTrafficDataService.java`). If the new CTS supports JSON, only the function `public static JSONObject getJsonObjectFromUrl(String url)` needs to be changed. The parsing of the object needs to be adjusted, too. For the *Show CTS Traffic Info Activity* (list view), this is done in the file `ShowCtsTrafficInfoActivity.java`, for the *Map View Activity* in `DriveAssistMapActivity.java`. If JSON is not supported, a new function, as well as parsing need to be implemented. The HTTP request can be reused nevertheless.

As it is probable that new message formats for the CAM and DENM will be standardised, it is possible to adjust the App to support new formats. All components which access a received CAM or DENM PDU need to be changed to fit to the new format. This must be done in the file `ReceiveMessagesServicePB.java` which implements the *Message Receiver Service*. In the *Database Adapter* (`DriveAssistDBAdapter.java`) both the functions `insertCamMsg(pbCamPdu _camPdu)` and `insertDenmMsg(pbDenmPdu _denmPdu)` have to be changed. Possibly the database tables have to be modified, if other fields need to be saved. For manually creating messages, the files `CreateCamMessageActivity.java` and `CreateDenmMessageActivity.java` have to be adjusted to build up the new message format.

Chapter 6.

Simulation of Traffic Scenarios

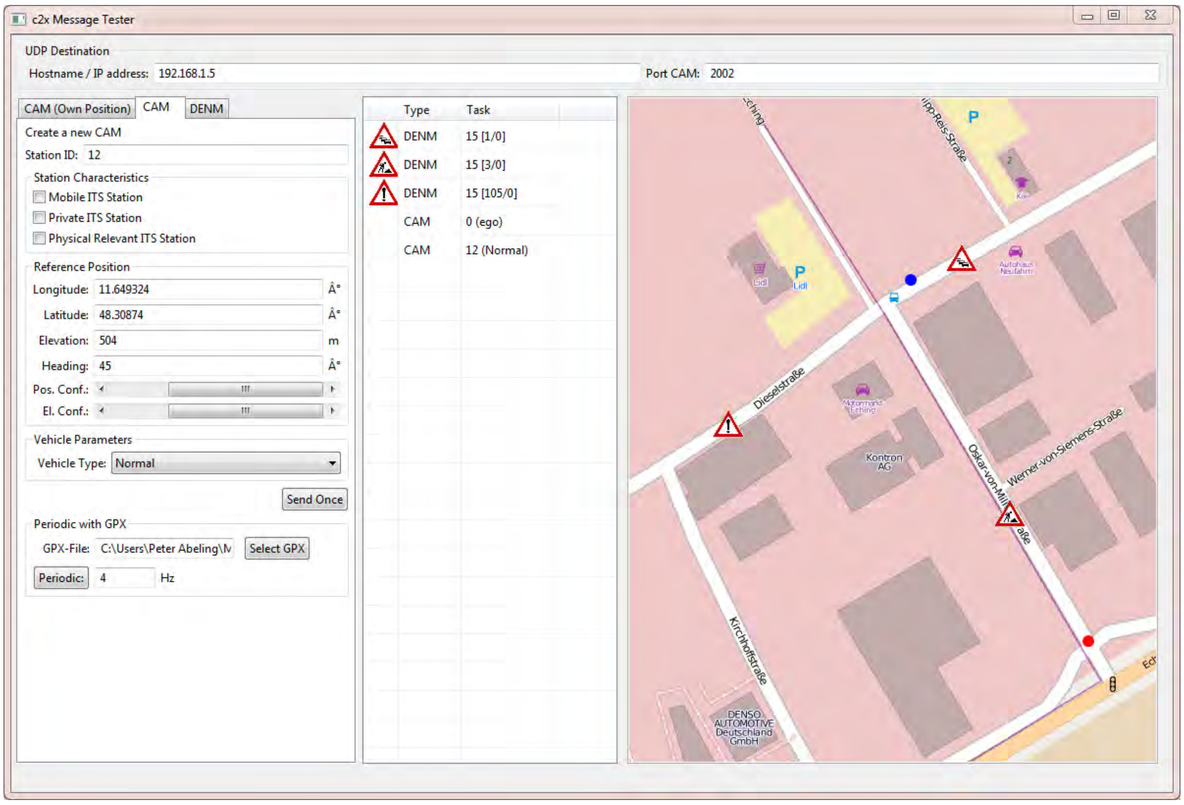
During the development process of the application, several simulations have been executed to test its functionality and performance before presenting it to test users. The development approach was incremental and iterative, i.e. whenever a new functionality had been added, it was tested and debugged immediately before continuing.

In the beginning, the basic communication between the simulation environment and the Android device was implemented. Afterwards, a simple user interface was added which was improved gradually throughout the whole development process. After defining the use cases and graphic renditions (warning screen and map view), the background services together with the warning screen were developed. Subsequently, the map view was implemented and, finally, all components which request and display Central Traffic Service (CTS) messages were added.

6.1. Simulation Environment: *c2xMessageTester*

The *c2xMessageTester* is a software for simulating Vehicle-to-X (V2X) communication scenarios as well as the user location (via an EGO Cooperative Awareness Message (CAM) mechanism). The Graphical User Interface (GUI) of the application is shown in Figure 6.1.

The software can create and send EGO CAMs, CAMs and DENMs which represent simulated traffic participants or events. Messages can either be emitted once or periodically with a predefined frequency. For creating messages, all obligatory fields have to be set. For moving events (EGO CAMs, CAMs) it is possible to follow pre-recorded GPS eXchange Format (GPX) tracks. Besides one instance of an EGO CAM sender, an unlimited number of other vehicles and incidents can be simulated. Active events or vehicles are shown in the list in the middle of the GUI. The user can follow the scenario on the provided map view on the right. Additionally, a position can be obtained directly out of the map by clicking on the desired position and pressing [shift] simultaneously. All message are sent to the User Datagram Protocol (UDP) destination specified in the top (Internet



c2x Message Tester

UDP Destination

Hostname / IP address: 192.168.1.5 Port CAM: 2002

CAM (Own Position) | CAM | DENM

Create a new CAM

Station ID: 12

Station Characteristics

- [ ] Mobile ITS Station
- [ ] Private ITS Station
- [ ] Physical Relevant ITS Station

Reference Position

Longitude: 11.649324 Â°

Latitude: 48.30874 Â°

Elevation: 504 m

Heading: 45 Â°

Pos. Conf.:

El. Conf.:

Vehicle Parameters

Vehicle Type: Normal

Send Once

Periodic with GPX

GPX-File: C:\Users\Peter Abeling\M Select GPX

Periodic: 4 Hz

Type	Task
DENM	15 [1/0]
DENM	15 [3/0]
DENM	15 [105/0]
CAM	0 (ego)
CAM	12 (Normal)

Figure 6.1.: *GUI of the c2xMessageTester* simulation tool. CAMs and DENMs can be generated and emitted once or periodically. Traffic scenarios can be monitored by using the integrated map view. Active events are shown on the list in the center.

Protocol (IP)-address and port). The tool allows to create complex V2X-based traffic scenarios and was used to test all implementation epochs of "Drive Assist".

6.2. Hardware Setup

Figure 6.2 shows the hardware setup for the simulation. The components Application Unit (AU) and Access Point (AP) are similar as in Figure 2.6 (Section 2.3). The AU receives V2X messages via a wireless network (Wireless Local Area Network (WLAN) IEEE 802.11n). The difference to the designated hardware setup (using the system in a car) is that all V2X messages are generated by the *c2xMessageTester* (SIM).

The simulation software encodes the messages before they are sent out via the AP. The application decodes the messages before they are accessible. This leads to a small performance degradation of the AU.

Using this simulation environment automatically creates some assumptions: No messages are lost

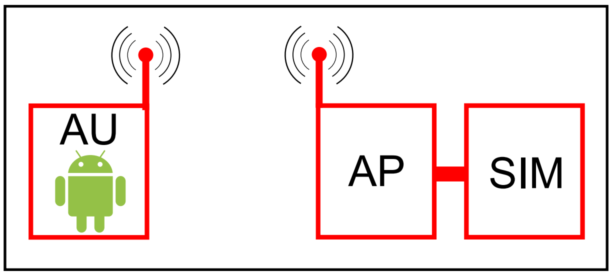


Figure 6.2.: Hardware setup for V2X communication simulation. The simulation environment (SIM) is represented by the *c2xMessageTester* tool. The tool creates V2X messages which are emitted via a connected Access Point (AP). The Application Unit (AU) uses a WLAN IEEE 802.11n connection to receive the messages.

or transmitted with errors. The real physical communication range which can differ strongly due to multipath or shadowing effects is substituted by a constant, adjustable range in the software. Therefore, all messages are always available. For future research, it should be considered to add more realistic constraints. For this research project the focus was set to develop or test an application under ideal conditions, as the key aspect is the user interface of the application.

6.3. Performance Related Issues

During the development process, the performance of the implementation was evaluated constantly. Two important fields have been figured out which strongly influence the performance of the application: The decoding of incoming messages on the device and the rendering of the map (together with bitmaps to display the user position or traffic events) are particularly time consuming. Details on both topics are given in this section.

The performance analyses have been done with the tool Dalvik Debug Monitoring Service (DDMS) (Section 4.4) which offers the possibility to profile all methods of the running application. The application was tested on a Samsung Galaxy Tab P1000[1] running Android 2.2. A scenario of one DENM-alerted event and the user position retrieved by an EGO CAM (both with $4\,\mathrm{Hz}$) have been shown on the map view to collect test samples. The binary serialisation library *Protocol Buffers* was used (also see Section 5.2.2).

6.3.1. Decoding of Messages

As incoming messages need to be decoded on the device, the performance of decoding messages strongly affects the application's total performance. The profiling trace shows, that the function

[1]Samsung Galaxy Tab GT-P1000, `http://www.samsung.com/at/consumer/mobile-phone/tablets/tablets/BGT-P1000`, last visited 3 March 2012.

`handleMessage(Message msg)` of the `Handler` object of the Message Receiver Service needs $20.109\,\mathrm{ms}$ to be executed (mean of 160 calls, including all child functions). The biggest contributor within the function is the decoding of both CAMs and DENMs. Decoding a DENM takes $15.756\,\mathrm{ms}$ (mean of 80 calls), decoding a CAM takes $7.988\,\mathrm{ms}$ (mean of 80 calls). DENMs are more complex to decode, because they contain more data fields and have deeper branching. A difference factor of approximately 2 for decoding both types was also obtained in other simulations done by the Distributed Multimodal Information Processing Group (VMI).

Another contributor is the implementation of the algorithm to manage decoded messages. The function `processDenmMsg(pbDenmPdu _denmPdu)` takes $5.521\,\mathrm{ms}$ to execute (mean of 80 calls), mostly in order to send a broadcast with event updates to the map ($4.082\,\mathrm{ms}$). The function `processCamMsg(pbCamPdu _camPdu)` needs $2.978\,\mathrm{ms}$ of processing time. In this performance test case, only the function `egoCamPositionUpdate(pbCamPdu _camPdu)` was called which also sends a broadcast.

In an earlier implementation, an Abstract Syntax Notation One (ASN.1) framework (for Java and C#) called *BinaryNotes* was used. Early tests showed that decoding with *BinaryNotes* considerably slowed down the application. Tests by the VMI showed that the finally chosen *Protocol Buffers* solution increases the decoding rate of CAMs by factor 400 and DENMs by factor 250 under ideal circumstances on the mobile device in comparison to *BinaryNotes*.

The calculations in this section show that there is still potential to further increase the performance of the application, even if the change from *BinaryNotes* to *Protocol Buffers* improved the decoding rate massively. Especially the warning management algorithms contain potential for performance optimisations.

6.3.2. Map Rendering

The complexity of rendering the map view and all related functions for displaying the user positions and traffic events have also been analysed. All numbers are obtained by calculating the mean value over 44 calls on the function. Drawing the map view (including all rendering of the map without overlays) takes $28.412\,\mathrm{ms}$. Besides, drawing the ego position on the map consumes with $27.335\,\mathrm{ms}$ nearly the same processing time. Broken down to its child functions, setting the center of the map to the user position is the biggest contributor ($20.335\,\mathrm{ms}$). Decoding and rotating the image for the user position consumes $6.107\,\mathrm{ms}$.

Depending on the number of active events on the map, it can be shown that the user position is not updated "smoothly" all the time. Sometimes the updates slightly lag, followed by some subsequent updates which are executed faster than intended. At the same time, the bitmap images request a lot of working memory for being drawn on the map. This caused an application crash in rare cases because Android manages memory allocation and cannot guarantee a sufficient amount of memory all the time. The reasons for both observations have to be studied in detail in further research.

Chapter 7.

Laboratory User Test

After the application *DriveAssist* has been implemented and tested, a laboratory user study has been designed and accomplished to evaluate whether or not the system fits the requirements concerning its functionality and usability. The central research question, together with some lead questions, was used to guide the study design. Test design and execution, as well as the results are described and discussed in this chapter.

7.1. Research Questions and Test Setup

In order to design the user study, a "Study Design Document" has been created as proposed by Brush [12] (Chapter 4, "Ubiquitous Computing Field Studies"). The document summarises all key facts concerning the organisation and execution of the study. It is available at the Distributed Multimodal Information Processing Group (VMI). Extracts of this document are presented in the following.

7.1.1. Research Questions

The central research question has been designed to verify whether the requirements defined in Section 5.1 have been achieved or not. The research question has been stated as in the following:

> Can an Android based visual information system using traffic information from **Vehicle-to-X (V2X) communication** and a **Central Traffic Service (CTS)** convince users of a significant increase of traffic safety and driving convenience when using such a system?

Additionally, some lead questions have been defined which concentrate on analysing special aspects of the system. The lead questions are:

- Is the User Interface (UI) easy to understand and to learn?

- Is Text-to-Speech (TTS) an appropriate way of supporting the warnings for the user?
- Would users prefer the *Map View Activity*, the *Warning Screen Activity* or both together in order to get informed about a traffic incident?
- What are strengths and weaknesses in the current implementation?

All questions in the questionnaire have been cross-checked against the research question and the lead questions in order to achieve the best possible interpretability and significance of the results.

7.1.2. Participant Profile

Most participating subjects work or study at the Technische Universität München. Those test users merely have an affinity to new technologies and are used to participate in user tests. Details on the subjects are given in Section 7.3.

7.1.3. Methodology and Conditions

The study was designed as a within-subjects study, i.e all participants took the same test track. Furthermore, the "Wizard of Oz"-technique was used [12]. The whole environment of test scenarios was simulated. This included the generation of the user position, other traffic participants and predefined traffic events. All data were generated by the tool *c2xMessageTester* which is presented in Section 6.1. For simulating CTS data, a static JavaScript Object Notation (JSON) object containg traffic information was stored locally on a server. The server was accessible from within the local network. Hence, every tester could access the same data.

For test data collection, a questionnaire has been used. It contained closed and open questions. Closed questions were yes-/no-questions, questions with given answers and questions in which a statement had to be rated on a Likert scale from "I totally agree", "I agree“, "neutral", "I disagree" to "I totally disagree". For answering open questions users had to write texts.

In some parts of the test, the subjects were given a distracting task in order to simulate driving activity. They were asked to follow a simulated scenario and continue playing the game *Superball* (see Figure 7.1) simultaneously[1]. In this game, the user controls a ball at the bottom of the screen, which can be positioned at the outer left and outer right of the grid. Other approaching balls must be avoided by shifting the user controlled ball to left and right. This task requires nearly all of the user's attention and is therefore comparable to the driving task with respect to the impact of attention to understand displayed warnings.

[1]The version of *Superball* which was used for the user test is a freeware developed by Christoph Stöpel, `http://christoph.stoepel.net/ViewSoftware.aspx?id=0103`, last visited 14 March 2012.

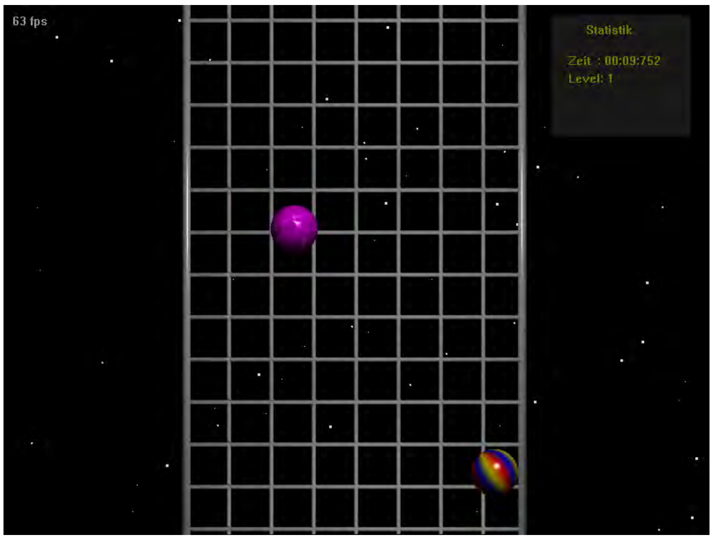


Figure 7.1.: *Superball* game screenshot. The user controls a ball (multicoloured) which can be shifted to left and right. The task is to avoid a collision with approaching balls (violet).

7.1.4. Test Hardware Setup

The hardware setup of the test is orientated on the simulation setup presented in Section 6.2. The test supervisor sits opposite to the test participant, both with a screen in front of them. The supervisor creates and starts simulated traffic scenarios using a version of the *c2xMessageTester*. The messages are sent to the test device by using an Access Point (AP) connected to the network. The user device uses a WiFi connection to the same network using the interface provided by this AP. On the test device (Samsung Galaxy Tab P1000 running Android 2.2), the latest version of *DriveAssist* is installed. The subject uses the screen and a keyboard in front of them to play the *Superball* game during chosen simulated scenarios.

Using this setup, the supervisor is able to start and stop the test scenarios considering the personal progress of every participant. Furthermore, the test users can be given support with difficulties or questions.

7.2. Test Execution

For participating in the user test, the test participants had to fill out a questionnaire. First, they had to read the introduction and to answer some introductory questions concerning their knowledge about similar systems. In the next part, the *Main Menu Activity* (Details in Section 5.2.1) was presented to them. When the subjects had looked at the main menu for some seconds, questions regarding their first impression were provided. In the following part, four different traffic scenarios were shown. While playing *Superball*, warning messages eligible to the scenarios were displayed by using the *Warning Screen Activity*. Afterwards the subjects were asked whether they had understood the situation. Those open questions were followed by general questions about the warning screen. For testing the *Map View Activity*, two scenarios were played by the supervisor and followed by the user. This time, the test users fully concentrated on the map without playing the game. For their opinion about the map view, several statements had to be rated afterwards.

For testing the usability of the Application (App), some tasks were provided to the users in the next part. While changing the Preferences or trying to display CTS traffic information in a list, the subjects were able to evaluate the usability in subsequent questions. Finally, the last part aimed at getting an overall impression from the subjects about the system. Here, they were asked explicitly to name strengths and weaknesses of the whole system. The mean test execution time has been 40 minutes.

7.3. Results and Interpretation

The results obtained by the laboratory user test are presented similarly to the organisation of the questionnaire. The interpretation of results is given directly in connection with the result demonstration in order to clarify the coherence. The full dataset with all test results is available at the VMI.

Section 7.3.1 concentrates on the previous knowledge of the subjects with respect to similar systems. Together with a demographic analysis of the subject pool, a basis for interpreting the test results is derived. In Section 7.3.2, the first impression of the subjects regarding the application is evaluated. All test users had never seen the application before which emphasises the significance of the derived test results in this section. The results for the warning management are presented in Section 7.3.3 and 7.3.4. Both the warning screen and the map view are evaluated, each in a separate section. In Section 7.3.5, all test results related to the usability of the application are presented and discussed. Finally, Section 7.3.6 copes with the overall impression of the subjects including an interpretation of strengths and weaknesses of the system.

7.3.1. Previous Knowledge with Similar Systems and Demography

A total number of twelve subjects participated in the study. The distribution of their ages is depicted in Figure 7.2. The average age of the subjects is 26.92 years and corresponds to the fact, that most participants were either students (five subjects) or research assistants (six subjects). Only one tester marked "other" as current status.

At the same time, ten out of twelve participants regularly use a smartphone. For evaluating the test, this fact must be taken into account because it means that the vast majority of test users is

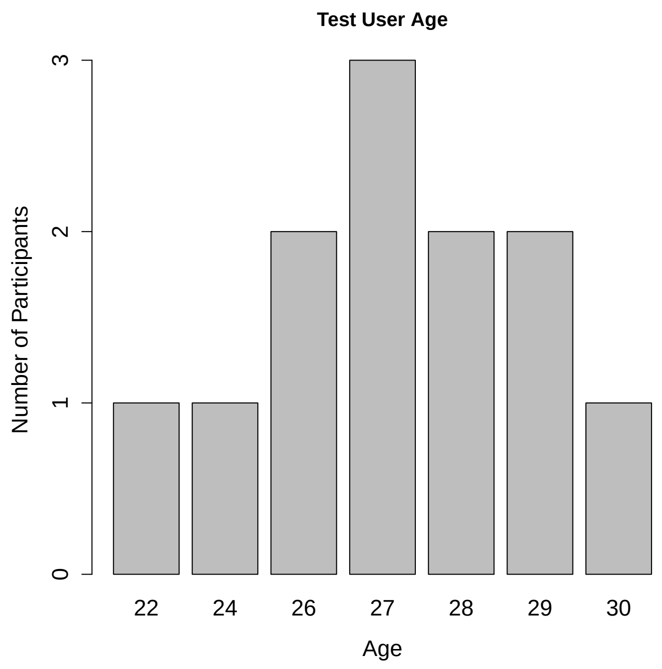


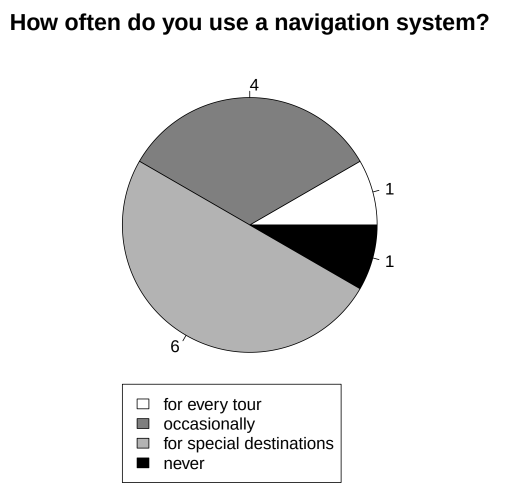
Figure 7.2.: Age distribution of participants in the laboratory user test.

used to utilise Apps in their everyday life. The results corresponding to the complexity of using the system are probably different (i.e. too optimistic) compared to a study with participants using a classic mobile phone or no phone at all.

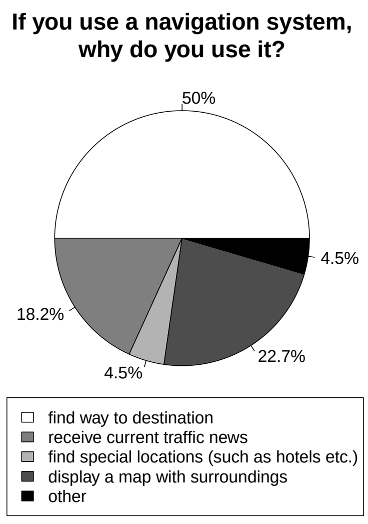
Eleven out of twelve subjects use a navigation system. Thus, the subjects know classic navigation functions and have previous knowledge in this field and also in using an aiding traffic system in the car. Their navigation system usage is shown in the diagrams in Figure 7.3. From Figure 7.3(a) follows that navigation systems are used by the subjects for specific reasons. The majority uses it occasionally (four subjects) or only to find special destinations (six subjects). This corresponds with the result in Figure 7.3(b). Asked for the use of their navigation systems, finding the way forms the top choice, followed by displaying a map with surroundings and receive current traffic news. It can be concluded that the need for receiving current traffic news and for observing the surroundings is present amongst the test participants, while finding the way is the main reason for using such a system.

Another question of the first part aimed at getting an impression of main traffic disturbances. The subjects answered as depicted in the pie chart in Figure 7.4.

From the viewpoint of the test users, a traffic congestion is the most disturbing traffic event (ten nominations, 37.04 %). Thereafter, risky overtaking of other traffic participants, bad weather conditions, street blockings and hazardous locations are nearly equally nominated. In the current



(a) Pie chart of answers showing that the majority of participants use navigation systems occasionally or for special destinations. The numbers indicate the number of participants who chose the corresponding answer.



(b) Pie chart of answers showing that finding the destination is the major use of navigation systems, followed by displaying a map with surroundings. Participants could tick multiple items.

Figure 7.3.: Navigation system usage of test participants.

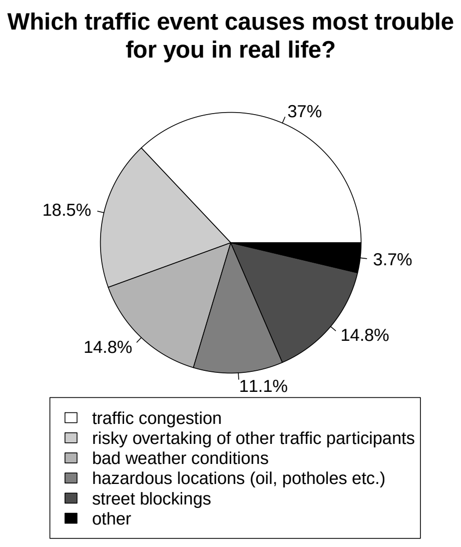


Figure 7.4.: Pie chart of answers showing that traffic congestions cause most trouble for the test participants in real life. Participants could tick multiple items.

version, *DriveAssist* supports warnings for traffic congestions and hazardous locations alerted by V2X communications. Warnings for bad weather conditions and street blockings are included by CTSs. The missing events could be integrated in future versions because it is envisioned to support them in V2X communication systems as well.

7.3.2. First Impression of *DriveAssist*

For evaluating the first impression of the application, the users had to look at the main menu of *DriveAssist* for several seconds before rating the statements concerning their first impression. The results are summarized in Table 7.1.

#	Statement	TA	A	N	D	TD
A1	The application seems to be complex.	0	1	1	7	3
A2	The application is graphically appealing to me.	0	7	2	3	0
A3	The yellow hint caught my attention immediately.	7	1	3	0	1
A4	The menu seems to be well usable in a car.	6	3	1	2	0
A5	The menu reminds me of my navigation system.	0	0	4	7	1

Table 7.1.: Number of nominations for a statement concerning the first impression on *DriveAssist*. The statements are rated from "I totally agree" (TA), "I agree" (A), "neutral" (N), "I disagree" (D) to "I totally disagree" (TD).

The majority of test users do not consider the application being complex (statement #A1). This result hints at the fact that users seem to be content with the clear and logic design of the main menu. In statement #A4, many users have valued the menu to be well usable in a car which was a requirement to the App (Section 5.1.1). At the same time, the main menu did not remind the subjects of their navigation system as visible in statement #5. The design of the user interface is based on navigation systems to ensure consistency, but this was not recognised by the test users. The reasons might be due to the fact that no image buttons and other colourful elements are used and that no button to enter a destination is available in the main menu. Following the results from statement #A2, for most subjects the menu seems graphically appealing (seven nominations for "I agree"). The yellow hint is a useful addition to the menu for the majority of subjects (statement #A3). All in all, the menu needs to be improved considering its graphical design to achieve a better score and address a broader range of taste. The first impression is important to convince users of buying such a system and in order to generate a feeling of attraction and interest in the application after the first glimpse [12].

Users were also asked about what they think is happening when pressing the buttons "Start Services" and "Traffic Info" in an open question. Ten out of twelve subjects answered wrongly in case of the "Start Services" button and five out of twelve in case of the "Traffic Info" button (including that vaguely correct answers were counted as correct ones). The result shows that the functionality of the buttons is unclear in the beginning, especially for the "Start Services" button. The term "Service" is probably too unspecified and should be replaced. It could also be considered to automatically start the Services when the application is started and not manually by pressing the button. This step might simplify the interaction.

7.3.3. Warning Screen Results

For testing and evaluating the *Warning Screen Activity*, four different pre-recorded scenarios have been played for every test participant. While playing *Superball*, warning screens have been displayed to the user to inform about what is happening around him (also see Section 7.1.3). After every scenario, the subjects had to answer the question "What do you think has happened in the situation in Scenario X?". The results to the open questions have been classified to the categories "The user has understood what has been happening" and "The user has not understood what has been happening". In doubtful cases, it was assumed that the subject has not understood the situation. This mechanism allows to evaluate the effectiveness of the warning screen with respect to comprehension speed and comprehension detail.

In test scenario 1, the user approaches a construction site $300\,\mathrm{m}$ ahead and passes it. Scenario 2 represents the situation that an emergency vehicle approaches from behind in $300\,\mathrm{m}$ with a higher speed and overtakes the user in the end. In scenario 3, another traffic participant sharply brakes in $200\,\mathrm{m}$ distance. The warning expires after $8\,\mathrm{s}$. Scenario 4 is more complex: The user approaches a traffic congestion in $300\,\mathrm{m}$. While he/she is passing by, a new warning evokes directly. It tells the user that a hazardous location is $150\,\mathrm{m}$ ahead. After having passed by the second event, a third warning evokes to inform that a stationary vehicle is $250\,\mathrm{m}$ ahead. The user finally also passes the stationary vehicle.

In all 4 scenarios, nearly all users understood the situation and were therefore able to understand the warning correctly. It is noticeable that in the first scenario three out of twelve did not understand the warning, but in the following scenarios only one (Scenario 2 and 4) or no one (Scenario 3) had problems to understand what was happening. As the complexity of the fourth scenario is higher than the others, it can be concluded that the users had quickly accustomed to the warning. This result confirms the effectiveness of the warning screen. Nevertheless, some suggestions for improvements are presented while evaluating details of the test results for this section.

The subjects again had to rate some predefined statements, this time considering the warning screen. The overall rating of every statement is summarised in Table 7.2.

The results of statement #B1 and #B2 confirm the impression mentioned above: users have quickly understood the warning and it was displayed right in time (and not violently interrupting as seen from statement #B6). The distance updates (statement #B4) also seem to be appropriate for the subjects. Four test users answered with "neutral" which might indicate that they do not care about the distance updates.

The indication of the direction is not considered useful in all cases. Nearly half of the test users marked "neutral" in statement #B3. Some subjects considered the direction indication as helpful, others did not. It is possible that some users were attracted by the game and found it difficult

#	Statement	TA	A	N	D	TD
B1	I have quickly understood the presented warnings.	7	4	0	1	0
B2	The information was shown too late.	0	0	3	6	3
B3	The indication of the direction was useful.	1	3	5	2	1
B4	The distance to the event is updated too slowly.	1	0	4	4	3
B5	Sometimes the red dot was "bouncing". That irritated me.	2	1	1	2	6
B6	The screen interrupted my attention violently.	0	0	3	8	1
B7	The voice output is a useful way of supporting the warning.	9	3	0	0	0
B8	The warning screen is ready to be used in a car.	2	6	2	2	0

Table 7.2.: Number of nominations for a statement concerning the *Warning Screen Activity*. The statements are rated from "I totally agree" (TA), "I agree" (A), "neutral" (N), "I disagree" (D) to "I totally disagree" (TD).

to pay attention and to interpret the red dot indicating the direction. The result for statement #B5 confirms this impression. The effect that the red dot sometimes "bounces" due to imprecise position updates (Section 5.2.6, "Updating the Direction") is not disturbing for a majority of subjects, but for some of them. Other approaches should be implemented and tested for direction indication in order to avoid misunderstandings. The indication of direction can in general be confirmed as being a useful support.

Using a TTS engine is considered very helpful because it focuses drivers on warnings when they are concentrated on driving tasks (statement #B7). All in all, most of the test users agreed with statement #B8, saying that the warning screen is ready to be used in a car. The proposed improvements for the direction indication could further increase the score for this statement, too.

The impression concerning the direction indication is confirmed in the comment section for the warning screen. Users stated that the warning screen is either not useful at all or only useful if the direction is shown on a continuous scale, such as a huge arrow moving constantly. Others indicated that four directions instead of eight are sufficient. Furthermore, it was remarked that the fonts should be bigger, and that the distance at which an event is displayed should depend on the driving speed.

7.3.4. Map View Results

The scenarios played for testing the *Map View Activity* are combinations of the scenarios used for the warning screen. Scenario 2 and 4 are combined to Scenario 5. Scenario 1 and 3 are combined to scenario 6. The test users followed scenario 5 and 6 on the map and rated the statements as shown in Table 7.3.

When looking at the results shown in Table 7.3, it can be seen that some statements have been

#	Statement	TA	A	N	D	TD
C1	It is distracting that the map in not turned, but the car.	1	3	0	4	4
C2	If a new warning is presented by the voice output, it should be clear where exactly it is.	6	3	3	0	0
C3	The overview the map provides over the surroundings is useful.	3	6	0	3	0
C4	It is easy to understand the meaning of the symbols.	4	6	1	1	0
C5	Only traffic events lying on my route should be displayed.	6	3	0	3	0
C6	The map view needs a lot of attention to understand what is happening.	0	4	5	1	2
C7	Newly detected events are presented in an appropriate way.	2	7	1	2	0
C8	The map view reminds me of my navigation system.	8	3	0	1	0
C9	Voice output is useful to support the map.	8	3	1	0	0
C10	The map should be turned, not the car.	3	1	5	1	2

Table 7.3.: Number of nominations for a statement concerning the *Map View Activity*. The statements are rated from “I totally agree” (TA), “I agree” (A), “neutral” (N), “I disagree” (D) to “I totally disagree” (TD).

rated unambiguously, others have not. From statement #C4, it can be concluded that the choice and design of the symbols is useful, as they are easily understandable for the majority of subjects. Voice output is considered useful (statement #C9) and the map view reminds many test users of their navigation system (statement #C8). The latter was intended to increase consistency compared with conventional navigation systems. User intuitively know what is happening because they recognise known patterns (in this case, the map and the user position symbol) [27].

Unlike in conventional navigation systems, *DriveAssist* does not rotate the map, but the map remains static while the car is rotated instead (the reasons are explained in Section 5.2.7). Eight out of twelve subjects marked that this unusual fact is not distracting (statement #C1). By asking statement #C10, the question was repeated negatively. This time, a majority of users marked “neutral” or “I disagree”/“I totally disagree”. Only four users thought that the map should be rotated under a fixed car. This is surprising because it is easier to put oneself in the driver’s position when the map is rotated (as it is more natural). The result should be checked with another implementation without using a change in perspective to confirm the advantages and disadvantages.

The map view offers a broad overview of the surroundings. The majority of test users evaluates this as useful in statement #C3. At the same time, it should be clear where a newly detected event is positioned (statement #C2) which is not always the case in the current implementation. An event can enter the “circle of interest” around the user and is presented as detected, even if it is not displayed on the map yet (because the zoom level is too deep). In future implementations, this should be considered and changed. A clear connection between a new event and its position should be guaranteed. Similar to the situation of the warning screen, the meaning of the symbols is easy to understand for most of the subjects (statement #C4). All the same, symbol comprehension could be improved by redesigning the symbols as many users marked "I agree" instead of "I totally agree".

The method of presenting new messages (TTS and symbol in the bottom left corner) is rather welcomed than declined in statement #C7. In the comment section of the map view part, some test users proposed to specify the type of the event in the audio output instead of generating the same sentence for all events. This could be included in future implementations.

Whether or not only traffic events lying on the driver's planned route should be displayed is a discussed question. Even if the majority agreed on this in statement #C5, three users disagreed. Considering the fact that the current implementation does not support the planning and navigation of a route, it could be included in future versions with navigation functionality. When adding such a function, the possibility to activate/deactivate "out-of-route" events in the Preferences should be included because some users seem to be interested in them as well.

Finally, the result for statement #C6 is very difficult to analyse. Whether the map view needs a lot of attention can not be fully clarified with this result and needs to be studied further.

7.3.5. Usability Results

In order to evaluate whether the usability requirements specified in Section 5.1.1 have been achieved or not, the test participants were given some simple exercises with *DriveAssist*. One tasks asked the subjects to change the Preferences and another task to display traffic information from CTSs in a list. Afterwards, the statements summarised in Table 7.4 were rated as depicted.

#	Statement	TA	A	N	D	TD
D1	It is easy to navigate through the application.	5	6	0	1	0
D2	I would prefer the portrait mode in comparison to the landscape mode.	0	1	2	5	4
D3	The preferences are too complex.	1	1	3	4	3
D4	The information provided by the hints are interesting.	1	4	2	3	2
D5	The preferences are easy to change.	4	5	2	1	0

Table 7.4.: Number of nominations for a statement concerning the application's usability. The statements are rated from "I totally agree" (TA), "I agree" (A), "neutral" (N), "I disagree" (D) to "I totally disagree" (TD).

The vast majority of test users confirms that it is easy to navigate through the application in statement #D1. At the same time, the Preferences are not considered to be too complex and could be changed easily by most of the subjects (statement #D3 and #D5). Both results were intended by the developer. The organisation of the *Preferences Activity* seems to have reached the goal to be clearly arranged and satisfactory commented.

The portrait mode as an alternative to the landscape mode (which was used for the laboratory user test) did not convince users (statement #D2). It offered no clear advantage compared to the landscape mode.

The information presented with the yellow hint which is displayed in the main menu must be discussed controversially. Five test participants totally disagreed or disagreed on statement #D4, whereas five other subjects agreed or totally agreed. It is likely that some users were bored by reading hints in text form in general and therefore did not want to be "flooded" with these kind of information. It must be analysed in further studies whether hints are useful at all and how they could be presented (design and content) to achieve a better score.

7.3.6. Overall Impression of *DriveAssist*

After having been introduced to a broad variety of functions of *DriveAssist*, the test participants were asked about their overall impression. To start with, some statements had to be rated. The result is shown in Table 7.5.

#	Statement	TA	A	N	D	TD
E1	The system has the potential to increase traffic safety.	5	6	0	1	0
E2	Using such a system in a car is very complicated.	0	0	2	9	1
E3	The functionality to navigate is missing.	4	7	1	0	0
E4	I would only use the map view and leave out the warning screen.	2	3	1	2	4
E5	Central traffic services are useful information to add in such a system.	8	3	1	0	0

Table 7.5.: Number of nominations for a statement concerning the overall impression about *Drive-Assist*. The statements are rated from "I totally agree" (TA), "I agree" (A), "neutral" (N), "I disagree" (D) to "I totally disagree" (TD).

The system is rated to be able to increase traffic safety which was the research question and major goal (statement #E1). The majority of subjects rated the system as easy to use (statement #E2) which was another goal when designing the UI. The third statement confirms an observation which was hinted at throughout the whole user test: The subjects clearly miss the functionality to navigate to a destination with *DriveAssist*. The current implementation is not understood as a separate system in addition to a navigation system, but as a substitute with enhanced functionality. Thus, in subsequent work on the App, a navigation module should be included.

The use of CTSs is highly appreciated by the test users (statement #E5) as a useful information to add. Especially when V2X services are not available, CTSs can be used to fill the information gap and to enrich and enhance the number of supported events.

Statement #E4 is more complex to analyse. The statement aims at comparing the user opinion about the *Warning Screen Activity* and the *Map View Activity*. A total number of five test users would not use the warning screen, but would only use the map view, as it is common for navigation systems. Technically, the warning screen offers the possibility to warn the user, even if other Activities or applications remain in the front. In these cases starting the map would require much more processing power. Hence, it would be more time-consuming and would decrease the immediacy of the warning which is an advantage of V2X-based warnings. The advantages of background services were not clearly presented to the subjects in the study. In a subsequent study, it should be analysed further while setting the focus on those facts. It is also thinkable to find another, new alternative to replace the warning screen with another mechanism.

The test participants were also explicitly asked about the most impressing function, weaknesses and strengths of the system. The following points could be classified from their answers:

Most impressing functions in the current implementation:

- Information about emergency vehicles or braking cars are accessible (which is not possible in conventional systems)
- The map view because it shows the surroundings of the user and supports a live view of moving objects
- The warning screen because it is clear and helpful
- The indication of the direction in the warning screen

Weaknesses of the current implementation:

- Graphical design: Fonts are partially too small, buttons should be replaced with images
- Navigation is not possible
- User position is not updated "smoothly" all the time on the map (it sometimes lags)
- Voice output could be more appealing and clearer
- Blue dot for an emergency vehicle on the map is mistakable

The strengths of the implementation have been evaluated as shown in Figure 7.5.

The results for the most impressing functions, weaknesses and strengths of the current implementation mirror the results from the previous sections. They show that the scope of the application is useful and that major functions (such as the warning screen and the map view) are clearly implemented and desired by the users. At the same time, the results show that the implementation

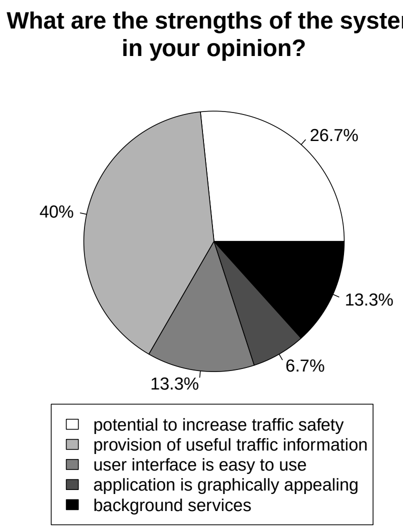


Figure 7.5.: Pie chart of answers showing the strengths of the application. Participants could tick multiple items.

should be improved, especially concerning the user interface, the indication of the direction on the warning screen and the performance of the application.

In the end of the questionnaire all subjects were asked if they could imagine to buy such a system in the future. Ten out of twelve subjects marked “Yes, if the function to navigate is included”, two marked “No”. The result strongly indicates again that users only want to use one single additional system in the car which is able to cover all desired tasks.

Asked for the price they would pay for an application such as *DriveAssist* (for example when available at the Android market), the majority marked “up to 50 €” (five nominations), followed by “it should be for free” (two nominations) and “up to 150 €” (two nominations). The result shows that users would invest in a system such as *DriveAssist* if it offers a variety of functions around the conventional functionality of a navigation system.

Chapter 8.

Conclusions

Visualising traffic information for assisting the driver is a highly relevant topic for successfully integrating Vehicle-to-X (V2X) communication services into the automotive domain. In the framework of this thesis, a driver assistance system named *DriveAssist* has been designed and implemented as an Android application. The system provides the user with traffic information from two different sources: it is able to receive and analyse messages from V2X communications (notably from Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs)) and generate warnings if appropriate. If activated, a message receiver service runs as a background thread all the time. Warnings can be presented to the user with a passive warning screen. This screen can also be evoked when another Activity or application is currently available on the Android device. Warnings can also be displayed centrally on the included map view. The messages are received over the device's WiFi unit and are decoded within the application. Other information sources are messages provided by Central Traffic Services (CTSs). This information is requested from a server by an Hypertext Transfer Protocol (HTTP) request, parsed and decoded. Afterwards, they can be displayed in a list view or on the map. The application is able to manage its own location using different methods (Global Positioning System (GPS), network provider, or EGO CAM). The application also provides a database for future implementations which are in need of received messages to analyse the past (e.g. for predicting potential collisions).

DriveAssist was tested during the development process by the developer and by test users in a laboratory user test. The simulations were aided by the software tool *c2xMessageTester* which is able to generate and send V2X messages from predefined scenarios with moving and static traffic participants. The tool can also generate the user position by sending EGO CAMs. During several tests, it could be derived that especially the decoding of the messages on the device, as well as the rendering of the elements on the map are particularly time consuming and performance relevant. Developing and implementing more efficient algorithms for the map management could further increase the system's performance.

The algorithms for managing the warnings using the warning screen or the map view are robust under the assumption that a certain traffic model can be applied. For more complex scenarios, these algorithms need to be adjusted. The structure of the application is modular which allows modifying or replacing certain components (for example to add another CTS source).

The laboratory user test showed that *DriveAssist* has the potential to increase traffic safety in the future. The test users were convinced of the core functionality and several implementation features, such as Text-to-Speech (TTS), the warning screen, and the map view. The potential of V2X communication has proven to likely convince users to invest in a such a system.

Even though, some adjustments concerning the application should be made in the future. A navigation module should be included. As users want to use one single system in their cars, one application should offer the whole range from navigation to V2X communication. CTSs are not considered to be a new feature, but one that is elaborated in current systems. The potential of V2X communication proved to be the most impressing feature (including the live view of other traffic participants) and should be accented. Furthermore, some user interface modifications and performance improvements are recommended to apply.

It is likely that applications running on mobile devices will become more and more important for the process of introducing V2X communication into the market. Mobile devices are already widespread and the high costs of integrating such systems in the car can be avoided. The market introduction can be pushed by adding other data coming from different sources (for example CTSs). At the same time, the driver context will be included more and more in the future. As already mentioned in combination with Floating Car Data (FCD) (Section 3.1), integrating car sensor data can be used to estimate the driving context. If the windscreen wiper is activated as well as the car lights, it probably rains and is dark outside. In this case warnings could be adapted to prevent an information overflow under difficult traffic situations. Modern applications could more and more use these techniques to further increase traffic safety and driving convenience.

Appendix A.

DriveAssist: Component Intent Tables

The communication between different components of the application *DriveAssist* is implemented using Android Intents. Intents are messages with a unique name which can carry additional information. They are broadcast throughout the whole application. Interested Activities can register a Broadcast Receiver to receive special Intents and extract their contents. This appendix presents tables for all Intents sent and received for all participating components corresponding to the communication diagram in Figure 5.1.

Location Service Intent Table

Intent Name	Intent Sender or Receiver	Description
Incoming Intents		
`INTENT_EGO_POS_UPDATE`	*Message Receiver Service*	EGO CAM user position update
Outgoing Intents		
`INTENT_LOCATION_UPDATE`	*Message Receiver Service*, *Warning Screen Activity*, *Map View Activity*, *Database Cleaner Service*	Current user position

Table A.1.: Location Service Intent Table

Message Receiver Service Intent Table

Intent Name	Intent Sender or Receiver	Description
Incoming Intents		
`INTENT_LOCATION_UPDATE`	*Location Service*	Current user position
`INTENT_WARNING_QUITTED`	*Warning Screen Activity*	Warning screen is closed
`INTENT_MAP_ACTIVE`	*Map View Activity*	Map view is started
`INTENT_MAP_INACTIVE`	*Map View Activity*	Map view is closed
`INTENT_TIMER_EVENT_REMOVAL`	*Map View Activity*	An event was removed, because it has expired
Outgoing Intents		
`INTENT_CAM_MSG_INSERTED`	No receivers	A new CAM has been inserted into the database
`INTENT_DENM_MSG_INSERTED`	No receivers	A new DENM has been inserted into the database
`INTENT_EGO_POS_UPDATE`	*Location Service*	EGO CAM user position update
`INTENT_UPDATE_EVENT_POS`	*Warning Screen Activity*	The active event has a new position
`INTENT_UPDATE_EVENT_POS_ON_MAP`	*Map View Activity*	An active event on the map has changed position
`INTENT_REMOVE_EVENT_FROM_MAP`	*Map View Activity*	An active event on the map is removed
`INTENT_RESET_WARNING`	*Warning Screen Activity*	The warning screen is reset with another event
`INTENT_KILL_ACTIVE_WARNING`	*Warning Screen Activity*	The active warning has to close itself

Table A.2.: Message Receiver Service Intent Table

For both Intents `INTENT_CAM_MSG_INSERTED` and `INTENT_DENM_MSG_INSERTED` no receiver is registered in the current implementation. They can be used in the future to implement efficient database access.

Request CTS Traffic Info Service Intent Table

Intent Name	Intent Sender or Receiver	Description
Incoming Intents		
`INTENT_REQUEST_TRAFFIC_INFO`	*Show CTS Traffic Info Activity*	Request a new JSON object with current traffic information
`INTENT_REQUEST_TRAFFIC_INFO_MAP`	*Map View Activity*	Request a new JSON object with current traffic information for the map view
Outgoing Intents		
`INTENT_NEW_CTS_TRAFFIC_INFO`	*Show CTS Traffic Info Activity*, *Map View Activity*	New JSON object containing current traffic information
`INTENT_CTS_REQUEST_ERROR`	*Show CTS Traffic Info Activity*, *Map View Activity*	The error which occurred when requesting new traffic data

Table A.3.: Request CTS Traffic Info Service Intent Table

Database Cleaner Service Intent Table

Intent Name	Intent Sender	Description
Incoming Intents		
`INTENT_LOCATION_UPDATE`	*Location Service*	Current user position

Table A.4.: Database Cleaner Service Intent Table

Warning Screen Activity Intent Table

Intent Name	Intent Sender or Receiver	Description
Incoming Intents		
`INTENT_RESET_WARNING`	*Message Receiver Service*	The warning screen is reset with another event
`INTENT_LOCATION_UPDATE`	*Location Service*	Current user position
`INTENT_UPDATE_EVENT_POS`	*Message Receiver Service*	The active event has a new position
`INTENT_KILL_ACTIVE_WARNING`	*Message Receiver Service*	The active warning has to close
Outgoing Intents		
`INTENT_WARNING_QUITTED`	*Message Receiver Service*	Warning screen is closed

Table A.5.: Warning Screen Activity Intent Table

Show CTS Traffic Info Activity Intent Table

Intent Name	Intent Sender or Receivers	Description
Incoming Intents		
`INTENT_NEW_CTS_TRAFFIC_INFO`	*Request CTS Traffic Info Service*	New JSON object containing current traffic information
`INTENT_CTS_REQUEST_ERROR`	*Request CTS Traffic Info Service*	The error which occurred when requesting new traffic data
Outgoing Intents		
`INTENT_REQUEST_TRAFFIC_INFO`	*Request CTS Traffic Info Service*	Request a new JSON object with current traffic information

Table A.6.: Show CTS Traffic Info Activity Intent Table

Map View Activity Intent Table

Intent Name	Intent Sender or Receiver	Description
Incoming Intents		
`INTENT_LOCATION_UPDATE`	*Location Service*	Current user position
`INTENT_REMOVE_EVENT_FROM_MAP`	*Message Receiver Service*	An active event on the map is removed
`INTENT_UPDATE_EVENT_POS_ON_MAP`	*Message Receiver Service*	An active event on the map has changed position
`INTENT_NEW_CTS_TRAFFIC_INFO`	*Request CTS Traffic Info Service*	New JSON object containing current traffic information
`INTENT_CTS_REQUEST_ERROR`	*Request CTS Traffic Info Service*	The error which occurred when requesting new traffic data
Outgoing Intents		
`INTENT_MAP_ACTIVE`	*Message Receiver Service*	Map Activity is started
`INTENT_MAP_INACTIVE`	*Message Receiver Service*	Map Activity is closed
`INTENT_TIMER_EVENT_REMOVAL`	*Message Receiver Service*	An event was removed, because it has expired
`INTENT_REQUEST_TRAFFIC_INFO_MAP`	*Request CTS Traffic Info Service*	Request a new JSON object with current traffic information for the map view

Table A.7.: Map View Activity Intent Table

Appendix B.

DriveAssist: Secondary Communication Diagram

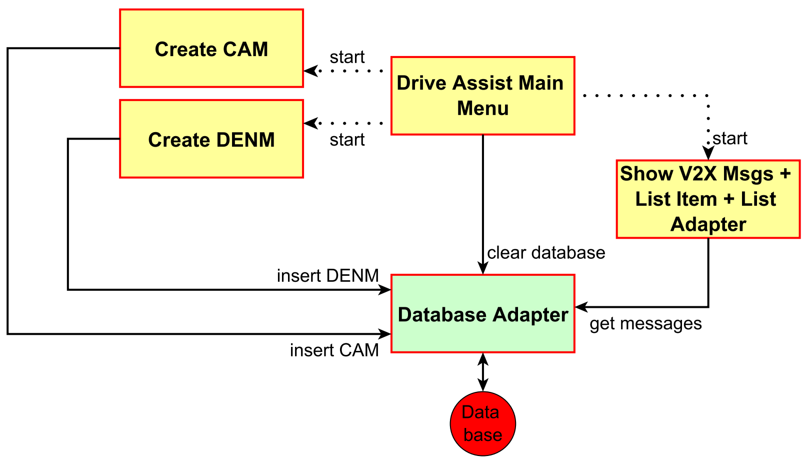


Figure B.1.: *DriveAssist* Secondary Communication Diagram. Yellow boxes symbolise Activities and blue boxes stand for helper classes. Red circles are input/output interfaces. Dotted lines concern starting and stopping of components. Solid lines represent all other communication.

Figure B.1 shows the secondary communication diagram for the application *DriveAssist*. When pressing the "menu" button of the device, both Activities for creating CAMs and DENMs can be started. After their creation, they are inserted in the database by using the *Database Adapter*. If the user calls "Show V2X messages" from the menu, the *Show V2X Messages Activity* is started. It requests all available messages from the database by also using the *Database Adapter*. To display all V2X messages in a list, a list adapter instance of `CamListArrayAdapter.java` or `DenmListArrayAdapter.java` respectively is used to display all list items. The result can be customised by setting filters. The filters can be set by pressing the devices' menu button in the *Show V2X Messages Activity*.

List of Figures

List of Tables

List of Acronyms

4G	4th Generation Mobile Telecommunications
ABS	Anti-lock Braking System
ADB	Android Debug Bridge
ADT	Android Developer Tools
AP	Access Point
API	Application Programming Interface
App	Application
ASN.1	Abstract Syntax Notation One
AU	Application Unit
C2CCC	Car-to-Car-Communication Consortium
CAM	Cooperative Awareness Message
CAN	Controller Area Network
CTS	Central Traffic Service
CU	Communication Unit
DDMS	Dalvik Debug Monitoring Service
DENM	Decentralized Environmental Notification Message
DRSC	Dedicated Short Range Communication
EEBL	Electronic Emergency Brake Lights
ESP	Electronic Stability Control
ETSI	European Telecommunications Standards Institute
FCD	Floating Car Data
FPD	Floating Phone Data

GN	GeoNetworking
GPRS	General Packet Radio Service
GPS	Global Positioning System
GPX	GPS eXchange Format
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
GW	Gateway
HMI	Human Machine Interface
HS	Hot Spot
HSDPA	High Speed Downlink Packet Access
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
ITS	Intelligent Transportation Systems
IVI	In-Vehicle Infotainment System
JDK	Java Development Kit
JSON	JavaScript Object Notation
NDK	Native Development Kit
NUI	Natural User Interface
OBD-II	On Board Diagnostics II
OBU	On-Board Unit
OHA	Open Handset Alliance
OSM	OpenStreetMap
PC	Personal Computer
PDU	Protocol Data Unit
PKI	Public Key Infrastructure
QoS	Quality of Service

RDS	Radio Data System
RSU	Roadside Unit
SD	Secure Digital
SDK	Software Development Kit
TMC	Traffic Message Channel
TS	Technical Specifications
TTS	Text-to-Speech
UDP	User Datagram Protocol
UI	User Interface
UMTS	Universal Mobile Telecommunications System
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-X
VANET	Vehicular Ad-hoc Network
VHF	Very High Frequency
VM	Virtual Machine
VMI	Distributed Multimodal Information Processing Group
WGS84	World Geodetic System 1984
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
XFCD	eXtended Floating Car Data
XML	Extensible Markup Language

Bibliography

[1] A. Mednis, G. Strazdins, R. Zviedris, G. Kanonirs, and L. Selavo, “Real Time Pothole Detection Using Android Smartphones with Accelerometers,” in *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, pp. 1 –6, June 2011.

[2] J. Zaldivar, C. Calafate, J. Cano, and P. Manzoni, “Providing Accident Detection in Vehicular Networks through OBD-II Devices and Android-Based Smartphones,” in *IEEE 36th Conference on Local Computer Networks (LCN), 2011*, pp. 813 –819, Oct. 2011.

[3] D. Kern, A. Schmidt, M. Pitz, and K. Bengler, “Status- und Kontextinformationen für die Telekommunikation im Auto,” in *Mensch & Computer* (T. Gross, ed.), pp. 119–128, Oldenbourg Verlag, 2007.

[4] M. Kranz, E. Weber, K. Frank, and D. H. Galceran, “Open Vehicular Data Interfaces for In-Car Context Inference,” in *AutomotiveUI ’09: Proceedings of the 1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, (New York, NY, USA), pp. 57–62, ACM, 2009.

[5] S. Diewald, A. Möller, L. Roalter, and M. Kranz, “Mobile Device Integration and Interaction in the Automotive Domain,” in *AutoNUI: Automotive Natural User Interfaces Workshop at the 3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI 2011)*, Nov.–Dec. 2011.

[6] D. K. Grimm, “Smartphone-Integrated Connectivity Applications for Vehicular Ad-hoc Networks,,” in *Proceedings of 18th ITS World Congress, 2011*, 2011.

[7] A. Lübke, “Car-to-Car Communication - Technologische Herausforderungen,” in *VDE Kongress 2004 Berlin - Innovationen für Menschen* (VDE, ed.), vol. 2, pp. 113–118, VDE, 2004.

[8] R. Baldessari, B. Boedekker, M. Deegener, A. Festag, W. Franz, C. C. Kellum, T. Kosch, A. Kovacs, M. Lenardi, C. Menig, T. Peichl, M. Röckl, D. Seeberger, M. Strassberger, H. Stratil, H.-J. Voegel, B. Weyl, and W. Zhang, “Car-2-Car Communication Consortium - Manifesto,” 2007.

[9] X. Yang, J. Liu, F. Zhao, and N. H. Vaidya, “A Vehicle-to-Vehicle Communication Pro-

tocol for Cooperative Collision Warning," *Annual International Conference on Mobile and Ubiquitous Systems*, pp. 114–123, 2004.

[10] V. Kharaev, "Comparative Analysis of Mobile Telecommunication Technologies for Intelligent Transport Systems," in *International Conference on Computational Technologies in Electrical and Electronics Engineering (SIBIRCON), 2010 IEEE Region 8*, pp. 294 –299, july 2010.

[11] A. Festag, H. Fußler, H. Hartenstein, A. Sarma, and R. Schmitz, "FLEETNET: Bringing Car-to-Car Communication into the Real World," *IEEE Computer*, vol. 4, no. L15, p. 16, 2004.

[12] J. Krumm, *Ubiquitous Computing Fundamentals*. Chapman & Hall/CRC, 1st ed., 2009.

[13] L. Barkuus and A. Dey, "Location-Based Services for Mobile Telephony: a Study of Users' Privacy Concerns," in *9th IFIP TC13 International Conference on Human-Computer Interaction, INTERACT 2003*, 2003.

[14] K. Matheus, R. Morich, and A. Lübke, "Economic Background of Car-to-Car Communication," *Proceedings of the 2. Braunschweiger Symposium Informationssysteme für mobile Anwendungen (IMA 2004), Braunschweig, Germany*, Oct. 2004.

[15] ETSI, *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 1: Functional Requirements (ETSI Technical Specification TS 102 637-1 V1.1.1)*. European Telecommunications Standards Institute, Sept. 2010.

[16] ETSI, *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specifications of Cooperative Awareness Basic Service (ETSI Technical Specification TS 102 637-2 V1.2.1)*. European Telecommunications Standards Institute, Mar. 2011.

[17] ITU, *SERIES X: Data Networks and Open System Communications; OSI Networking and System Aspects - Abstract Syntax Notation One (ASN.1) (ITU-T Recommendation X.691)*. International Telecommunications Union, July 2002.

[18] ETSI, *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service (ETSI Technical Specification TS 102 637-3 V1.1.1)*. European Telecommunications Standards Institute, Sept. 2010.

[19] M. Röckl, K. Frank, P. Robertson, and T. Strang, "Enhancing Driver Assistance Systems by Cooperative Situation Awareness," in *DGON-Symposium: System Verkehr - Steuern, Regeln, Entwickeln* (D. G. fuer Ortung und Navigation e.V. (DGON), ed.), 11 2006.

[20] M. Green, "How Long Does It Take to Stop? Methodological Analysis of Driver Perception-Brake Times," *Transportation Human Factors*, vol. 2, no. 3, pp. 195–216, 2000.

[21] W. Huber, M. Lädke, and R. Ogger, "Extended Floating-Car Data for the Acquisition of Traffic Information," in *Proceedings of the 6th World Congress on Intelligent Transport Systems*, 1999.

[22] M. Friedrich, P. Jehlicka, T. Otterstätter, and J. Schlaich, "Mobile Phone Data for Telematic Applications," in *Proceedings of International Multi-Conference on Engineering and Technological Innovation: IMETI 2008*, (International Institute of Informatics and Systemics (IIIS), Orlando, Florida, USA), 2008.

[23] TomTom, "TomTom White Paper: How TomTom's HD Traffic (TM) and IQ Routes (TM) Data Provides the Very Best Routing," tech. rep., 2007.

[24] R. Meier, *Professional Android 2 Application Development*. Birmingham, UK, UK: Wrox Press Ltd., 1st ed., 2010.

[25] A. J. McKnight and A. S. McKnight, "The Effect of Cellular Phone Use upon Driver Attention," *Accident Analysis and Prevention*, vol. 25, no. 3, pp. 259 – 265, 1993.

[26] J. Sonnenberg, "Service and User Interface Transfer from Nomadic Devices to Car Infotainment Systems," in *Proceedings of the 2nd International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '10, (New York, NY, USA), pp. 162–165, ACM, 2010.

[27] B. Shneiderman and C. Plaisant, *Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th Edition)*. Addison Wesley, 4 ed., Apr. 2004.

[28] R. W. Sinnott, "Virtues of the Haversine," *Sky and Telescope*, vol. 68, no. 2, pp. 158 – 159, 1984.