Designing and Implementing a GPS-based Vehicle Navigation Application for Eclipse Kuksa

University of Oulu
Department of Information Processing Science
Master's Thesis
Zhihong Wu
16/12/2020

Abstract

With the development of the Internet of Things (IoT), connected cars are rapidly becoming an essential milestone in the design of intelligent transportation systems and a key element in smart city design. Connected cars use a three-layer client-connection-cloud architecture, and car sensors are located at the client layer. This architecture provides the driver with a large amount of data about the external environment, which reduces the number of traffic accidents and helps the car drive safely. Driving safety is the most critical design factor for next-generation vehicles. The future vision of the automotive industry is self-driving cars. However, it faces some challenges.

Eclipse Kuksa provides solutions to challenges in the field of connected cars. A comprehensive ecosystem includes a complete tool stack for connected vehicles, including a vehicle platform, a cloud platform, and an application development Integrated Development Environment (IDE). Its essential function is to collect, store, and analyze vehicle data and transmit various information in the cloud.

This master's thesis aims to investigate a Global Positioning System (GPS) -based vehicle navigation application on the vehicle and cloud platforms of Eclipse Kuksa, understand how to develop a GPS-based vehicle navigation application using the Eclipse Kuksa software platform, and discuss the advantages and challenges of using Eclipse Kuksa to develop vehicle applications. The research methods are Design Science Research (DSR) and literature review. System development is carried out following the Design Science Research Methodology (DSRM) Process, developed and evaluated on the vehicle navigation application. The application artifact consists of the Eclipse Kuksa vehicle platform and cloud platform. The steps described in this paper can be used to build vehicle applications in Eclipse Kuksa. This paper also explains the benefits and challenges of using Eclipse Kuksa to develop vehicle applications. The main benefit is that open source solutions break the long-term closed development model of the automotive industry and establish a vehicle-to-cloud solution standard to meet the IoT challenges to the automotive industry. Simultaneously the challenge of using Eclipse Kuksa is the complexity of environment construction and the software and hardware compatibility.

Keywords
V2X, IoT, 5G, Navigation Application, Eclipse Kuksa, Open Source, Connected Car, Kuksa ecosystem, Automotive System

Supervisor
Doctor of philosophy, Teemu Karvonen
Master of Science, Arun Sojan Kudakacheril

Foreword

I want to take this opportunity to sincerely thank professor Pasi Kuvaja for introducing me to this project, which has taught me a lot about the automotive industry.

I want to express my deep gratitude to my supervisors, Dr Teemu Kaorvonen and Arun Sojan Kudakacheril, for their support and encouragement to help me complete this thesis. I would like to thank Dr Markus Kelanti for his valuable feedback to help me, also thank Ahmad Banijamali and other M3S members at the University of Oulu for their help with this thesis.

Finally, I would like to thank my family and friends who always support me and believe in me, especially during difficult times.

Contents

1. Introduction

Since the introduction of electronic systems into vehicles in the ‘60s, the automotive industry has undergone tremendous changes, and software has become the primary source of automotive innovation (Mössinger, 2010). The initial research and development of the automotive industry lasted about 30 years, from 1966 to 1995. There were good ideas during that time, but they were impossible to implement due to the lack of technology. Later, from 1995 to 2002, embedded modules were used in the car, which then, from 2007 to 2012, introduced the vehicle’s information and entertainment applications. Third-party applications and software providers joined the ecosystem, after which the automotive industry entered the connected car V2X (Vehicle-to-everything) era in 2012. (Krasniqi & Hajrizi, 2016.) “Connected Cars” is a term for vehicles with internet access that allows devices inside and outside the car to interact with each other, establish an internet connection with the back-end application and share various data capabilities. Data is transferred to the back-end service, and multiple workflows can be created to take the necessary actions (Dhall & Solanki, 2017). V2X communication allows vehicle communication with any vehicle or infrastructure. There are devices such as sensors and on-board computers in the car. Connected cars rely on sensors and related technologies to communicate with other vehicles and infrastructure (Buehler, 2018). They can link to smartphones, provide emergency roadside assistance, register real-time traffic alerts, etc. (Krasniqi & Hajrizi, 2016). They can also send alarms to each other under certain circumstances, such as collisions that may occur (Dhall & Solanki, 2017). One of the main goals is to improve road safety and reduce traffic accidents (Golestan, Soua, Karray, & Kamel, 2016). The Internet of Things (IoT) is an emerging technology. It consists of two words: “Internet” and “Things”. The first word “Internet” refers to a global system composed of interconnected computer networks that use a standard internet protocol suite TCP/IP (Transmission Control Protocol/Internet Protocol) to provide services to billions of users worldwide, and the word “Things” can be any object or person distinguished by the real world. Combining these two words reveals that the IoT provides a unique identification for each object. The sensors and actuators embedded in the objects are linked through wired and wireless networks and connected to the internet through internet IP. IoT allows connection and communication between human-to-human, human-to-things, and things-to-things, which is anything in the world. (Madakam, Ramaswamy, & Tripathi, 2015.) With the rise of the IoT, the rapid development of IoT technology, the customer is willing to wait for must-have technical functions, connectivity, and ease of use of cars (Ayres, 2018). These factors drove the automotive industry’s transition from the product age to the service and experience era (Krasniqi & Hajrizi, 2016). The connected car utilizes a three-layer client-connection-cloud architecture. All sensors are located on the client layer. These sensors are used to collect data, also import-related data through the network communication connection between the car and the vehicle provided by the connection layer. The connection layer handles the different wireless communication types on connected cars to ensure connectivity to existing networks. The cloud layer uploads data storage, processing and analysis. However, connected cars use a large amount of data and information pools, the diversity and massive scale of available data and information are the main challenges facing connected cars. (Golestan et al., 2016.) Original Equipment Manufacturer (OEM)s deploy to address challenges by creating and managing IT professionals, software developers, and engineering teams (Ayres, 2018).

According to About Eclipse-Kuksa (2018), Eclipse Kuksa provides technology in the field of the V2X environment. With platforms for (a) in-vehicle, (b) cloud, and (c)

connected vehicle application development, a solution to the challenge is provided: a comprehensive ecosystem.

This ecosystem offers:

- a in-vehicle software platform,
- a cloud software platform and
- an application development Integrated Development Environment (IDE) with an inclusive environment across a variety of frameworks and technologies,

It is a complete tool stack for the connected car domain.

The essential functions of this environment are collecting, storing, and analyzing vehicle data in the cloud and transferring various pieces of information, such as cloud calculation results (e.g., improved routing), software maintenance updates, or even brand new applications. Simultaneously, while considering current emerging technologies like IoT, 5G connectivity, bringing vehicles, IoT, cloud, and security technology is together to facilitate the development of suppliers, OEMs, enterprises, and developers. The stakeholder of the Eclipse Kuksa project is APPSTACLE project (stands for open standard APplication Platform for carS and TrAnsportation vehiCLEs). (About Kuksa, 2018; About Eclipse-Kuksa, 2018.) The APPSTACLE project brings together several leading automotive software and telecommunications companies and research institutions, intending to simplify the automotive system development process through an open and secure cloud platform (Pakanen et al., 2017).

In the 1990s, the 802.11 wireless standard protocol enabled communication between vehicles driving on the road, the car, and the intelligent road infrastructure. Also, it deployed the Global Positioning System (GPS) in the car. These formed the basis of the smart transportation system. A vehicle is considered a moving node and can track its location to perform traffic management operations. Real-time navigation and road information services can provide drivers with safer and more reasonable driving. Navigation supplies the route planning; the user plans a route at home and sends it to the car to view nearby places' photos and suggestions. Simultaneously, the navigation system uses dynamic real-time traffic information and can also provide the best possible route to the users and give dynamic recommendations to the driver (Coppola & Morisio, 2016). Eclipse Kuksa provides a Rover tool, a mobile robot, more like a mini car that can demonstrate the vehicle connects to the cloud through the network communication connection. This master's thesis's motivation chooses to develop a GPS-based vehicle navigation application in Eclipse Kuksa to validate Eclipse Kuksa utility for the development of the V2X applications. The GPS-based vehicle navigation application includes two main functional modules:

1. As the Rover tool hasn't yet had the GPS module, this thesis implementation uses mock GPS, simulating the vehicle's location information. The application obtains vehicle location information from a recorded GPS file in the in-vehicle system, and the vehicle location information is uploaded to the cloud,

2. The cloud application receives the vehicle location information sent from the in-vehicle system and then displays it graphically. The vehicle location information is also saved in the database.

The evaluation of this GPS-based vehicle navigation application demonstrates the utility of the development using Eclipse Kuksa. It can help develop vehicle projects based on

Eclipse Kuksa. Also, the results can be used as a foundation for further extending the Eclipse Kuksa. Therefore, there are two research questions:

RQ1: How to develop a GPS-based vehicle navigation application using the Eclipse Kuksa software platform?

RQ2: What are the advantages and challenges of developing a vehicle application using the Eclipse Kuksa software platform?

Design science research (DSR) was used as the research method for this thesis to answer the research questions. The DSR methodology (DSRM) process was used as the design cycle for developing the GPS-based vehicle navigation application. The main contribution was the general application to demonstrate the process to create an app through the in-vehicle system to the cloud and from the cloud to the client using the Eclipse Kuksa software platform. It validated the utility of the development of V2X (Vehicle-to-everything) applications using Eclipse Kuksa. The developed application also supported the further expansion of other vehicle projects in the Eclipse Kuksa software platform. This master's thesis research was supported by M3S research unit of Oulu University.

The paper proceeds as follows. First, Chapter 2 defines the research objectives and questions and describes the research methods of DSR and the DSRM process. The related techniques, literature review collection the knowledge of previous research, and the background of Eclipse Kuksa are introduced in Chapter 3. Chapter 4 follows the DSRM process from the requirement for the objective-centered solution startup descriptions, design, development, demonstration, and evaluation to implement the GPS-based vehicle navigation application. The results of the technical literature in Chapter 3 are used here, too. The research questions and results are discussed in Chapter 5. Finally, Chapter 6 summarizes the thesis and its limitations and makes a proposal for future research.

2. Research Methodology

This chapter consists of two parts. The first part introduces the research objective and research questions. In the second part, DSR is described as a research method, and the DSRM process is explained as the action of the research. The first part includes determining the research objective. The reasons for choosing the research method and all phases of the DSRM process are included in the second part.

2.1 Research Objective and Research Questions

Eclipse Kuksa supplies a novel development platform. The structure of the platform had been set up, and the system was complex. The author was curious about what the development environment provided by Eclipse Kuksa was like for developers. Consequently, the research goal was to develop an application to validate the Eclipse Kuksa development environment's utility. Literature reviews of the automotive industry software development had been carried out to gather prior knowledge about research topics. Those reviews focused on the domain of connected cars and the development of vehicle tracking systems and the global navigation satellite system's information, including GPS and the connected car. After applying the relevant information on the literature research topic and combined with the development environment information provided by Eclipse Kuksa, it was analyzed that the Eclipse Kuksa system did not include a GPS receiver module. Through this research, the Eclipse Kuksa system's limitations were supplemented, and finally, the study of developing a mock GPS adopted vehicle navigation application was determined. The more explicit research objective was therefore established:

- *To investigate a GPS-based vehicle navigation application on the in-vehicle and cloud platform of Eclipse Kuksa.*

The research objective generated the following two research questions:

- *RQ1: How to develop a GPS-based vehicle navigation application using the Eclipse Kuksa software platform?*
- *RQ2: What are the advantages and challenges of developing a vehicle application using the Eclipse Kuksa software platform?*

The first research question (RQ1) was the leading research question. It was to find a development process for a GPS-based vehicle navigation application in the development environment provided by Eclipse Kuksa. The application was suitable for the driver to navigate the vehicle in real-time while simultaneously adapting the fleet manager's real-time vehicle location monitoring and management. At the same time, it demonstrated to other developers who develop the application using the Eclipse Kuksa.

The second research question (RQ2) is a sub-research question of RQ1. RQ2 focused on evaluating the development environment provided by the Eclipse Kuksa used by the application. The advantages and disadvantages of the development environment provided by the Eclipse Kuksa were defined as evaluation results. The evaluation of the results contributed to improvements in certain aspects of the Eclipse Kuksa. On the other hand, other developers could choose to develop in Eclipse Kuksa based on the advantages and challenges described in this paper.

2.2 Research Method

DSR is a research paradigm that develops innovative and useful artifacts to answer questions to human problems and produce new design knowledge to scientific evidence. The design science paradigm is fundamentally a problem-solving paradigm, and its final target is to make an artifact that is constructed and then evaluated. The artifact is a term used to describe something that humans build. (Hevner & Chatterjee, 2010). Therefore, DSR was used in this thesis to develop a solution using Eclipse Kuksa. DSR emphasizes the utility of innovation, and the goal of DSR is a utility (Järvinen, 2004). This thesis used DSR to validate the Eclipse Kuksa development environment's utility by creating a GPS-based vehicle navigation application in Eclipse Kuksa. The artifact of this research was the vehicle navigation application. The DSR cycles in Figure 1 are used as the basis of the information system research framework (Hevner, 2007). They need to be presented and identified in a DSR project (Hevner & Chatterjee, 2010).

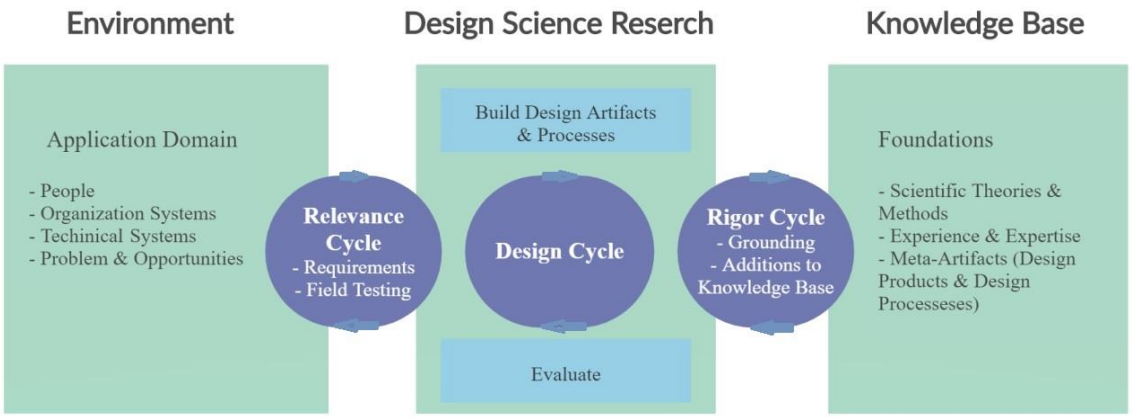


Figure 1. Design Science Research (DSR) cycles (adapted from Hevner, 2007)

2.2.1 Three inherent research cycles of DSR

The DSR cycles are three closely related activity cycles. The goal is to improve the environment by introducing new artifacts and the process of building these artifacts. (Hevner and Chatterjee, 2010). Through these three cycles to complete it (Hevner, 2007).

Relevance Cycle

The relevance cycle links the research project's environment to the DSR activities. The application domain in the environment contains people, organization systems, and technical systems, interacting with the actual research work in the DSR activities to achieve the goals. The relevance cycle starts DSR with a special context. It does not only input the requirements where the problem/opportunity evolves in the real application environment into the research. It also defines the introduction of the research artifact into the environmental field testing as an output. This output is the acceptance criteria for an evaluation in the application environment. (Hevner & Chatterjee, 2010).

Rigor Cycle

The rigor cycle combines the DSR activities with a knowledge base. The knowledge base provides fundamental scientific theories and methods, experience and expertise, and the existing artifacts and processes in the application domain. The rigor cycle extracts

information about the research project from the knowledge base; the whole research iteration experience is added to the knowledge base. It does this either by adding original theories and methods into the research process or creating new artifacts. The latter involves designing products and processes (Hevner & Chatterjee, 2010).

Design Cycle

The design cycle iterates between the construction of the artifact and its evaluation to further refine the design. It is the core of a DSR project. This phase designs the artifacts and evaluates them according to the requirements until a satisfactory design is achieved (Hevner & Chatterjee, 2010).

To conclude, the requirements are inputted from the relevance cycle, and the design and evaluation theory and methods are derived from the rigor cycle. The design cycle is a vital part of the completion of DSR. The successful completion of the design cycle depends on two other cycles. Still, at the same time, the design cycle is relatively independent in the actual implementation of the research project (Hevner & Chatterjee, 2010). This thesis was about developing a GPS-based vehicle navigation application in Eclipse Kuksa. The technical environment was Eclipse Kuksa; people were the researchers involved in the Eclipse Kuksa project. The organization system was related to the Eclipse Kuksa project organization. This thesis guidance was from the knowledge base, including the research method, designing, and implementation process. The literature review guided the development and implementation of GPS-based vehicle navigation applications, and the Eclipse Kuksa provided the development environment. Furthermore, this research project's design cycle through the DSRM process model was a recognized framework to introduce research in Figure 2 (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007).

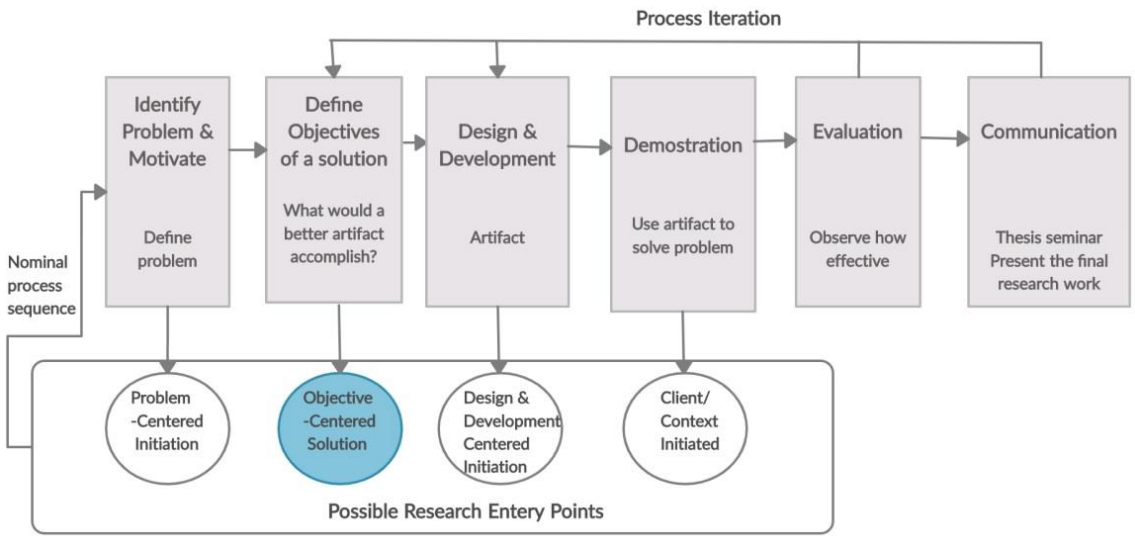


Figure 2. DSRM process model (adapted from Peffers et al., 2007)

2.2.2 DSRM Process Model

As shown in Figure 2, the design science process includes six steps: identify problem and motivation, define objectives of a solution, design and development, demonstration, evaluation, and communication. The possible research entry points can be problem-

centered initiation, objective-centered solution, design and development centered initiation, or client/context initiated (Peffers et al., 2007). This DSRM process model provides a road map for this thesis.

The research entry point of this thesis was the objective-centered solution. As described in 2.1, this thesis first established the research objective and then formulated the two research questions. Therefore, the process sequence of this research began from step 2 – defining the objectives of a solution.

Define objectives of a solution

Peffers et al. (2007) developed the solution's goals from the problem definition and possible and feasible knowledge. This thesis analysed and combined the knowledge obtained by literature reviews to put forward a software requirement specification, which refers to requirements in this paper, to meet the problem solution. According to Bassil (2012), the software requirement specification is a complete and overall description of the under-developed software's behavior. It includes functional and non-functional requirements. Functional requirements generally describe the interaction between users and software, including requirements such as functions, software attributes, interface requirements, and database requirements. Non-functional requirements refer to various standards, constraints, and restrictions imposed on the software's design and operation. It includes such as reliability, scalability, availability, maintainability, performance, and so on. This thesis requirement had functional and non-functional requirements to provide a complete and overall description of the GPS-based vehicle navigation application's behavior under consideration. The Eclipse Kuksa environment and literature reviews on connected cars, car navigation system and vehicle tracking system were used to find a solution.

Design and development

Referring to Peffers et al. (2007), this is the process of creating artifacts, including determining the required functionality and architecture of the artifacts. The contribution of the research is embedded in the design. This thesis was developing a GPS-based vehicle navigation application in which design and development activities were further divided into two separate parts to describe. The design activity is the process of defining the plan for a solution, including software architecture design, database design, and graphical user interface design (Bassil, 2012). This thesis's design phase was designing the GPS-based vehicle navigation application's software solution, including system architecture design of the vehicle navigation application and MQTT architecture design. The development phase is the process of realizing the requirement and design into production; According to the requirement and design specification, the program and code are written (Bassil, 2012). This thesis was programming with two separate applications, an in-vehicle application, and a cloud application. The related files were created according to the requirement and architecture design in the process.

Demonstration

Peffers et al. (2007) figure out that demonstrate one or more examples of using artifacts to solve a problem. The artifact of this thesis was the GPS-based vehicle navigation application. Therefore, this thesis executed the GPS-based vehicle navigation application to illustrate the solution, including an in-vehicle application was running on the in-vehicle platform and a cloud application was running on the cloud platform.

Evaluation

Referring to Peffers et al. (2007), evaluation is about test how well the artifacts support a solution to solve the problem. This activity involves comparing the solution's goals with actual observations of the artifacts used in the demonstration. This thesis aimed to examine the implementation and running results of the GPS-based vehicle navigation application used in the demonstration to meet the pre-defined functional and non-functional requirements. According to Harrold (2000), testing is a necessary process carried out to support quality assurance. Quality assurance assists in ensuring the development of high-quality software. The testing includes designing test cases, using these test cases to execute software, and checking the executing results. This thesis evaluated the GPS-based vehicle navigation application's implementation through testing, designed test cases according to each pre-defined functional and non-functional requirement, executed test cases, and checked execution results.

Communication

According to Peffers et al. (2007), there is a need to communicate to spread the generated knowledge. Communication had been handled by participating in the master's thesis seminar and by presenting the final research work.

3. Literature Review and Background

This chapter explains the literature review for the main concepts around connected car, car navigation system, and vehicle tracking system. It also explains the background of Eclipse Kuksa and the related Rover tool used for developing the vehicle navigation application. The first section presents the literature review of the car navigation system and vehicle tracking system, both of which are related to GPS. The second section describes the background of Eclipse Kuksa.

3.1 Literature Review

The main research question of this paper was how to develop a GPS-based vehicle navigation application using the Eclipse Kuksa software platform. The literature review was used to obtain the necessary theoretical insights and design methods for building GPS-based vehicle navigation applications. A search was conducted to find the relevant literature for this, starting from the connected car domain. Google Scholar tool was used to search for keywords such as connected car, automated car, self-driving car, autonomous car, IoT, Internet of Things, GPS navigation, car navigation system, real-time vehicle tracking system, connected vehicle challenge and automotive industry. The preliminary screening was conducted on abstracts of relevant scientific articles. The introduction and conclusion were proceeded for further filtering and then the full paper to be obtained the final shortlist of relevant literature. Google was adopted to search for the latest information and statistics. The following sections present the literature review on connected cars, car navigation systems and the vehicle tracking system.

3.1.1 Connected Car

A connected car is a term used to describe a vehicle can access the internet and communicate with smart devices inside the car itself, in other cars and in the road infrastructure. It can collect real-time data from multiple sources. (Dhall & Solanki, 2017; Coppola & Morisio, 2016.)

The first connected car was co-developed by General Motors and Motorola Automotive on OnStar in 1996. OnStar is mainly a General Motors based car that provides safety information services. When a car accident occurs, a call centre deployed with airbags is used, which has a voice call function to contact emergency responders. Many automakers followed suit, connecting cars to emergency responders. (Auto Connected Car, 2014.) Later, with the development of information technology, Wireless Local Area Network (WLAN) protocol-IEEE802.11 began to enable the communication between different vehicles on the road and between cars and the road infrastructures (Coppola & Morisio, 2016). GPS locations and the features of both voice and data were introduced into the safety system, adding capabilities of remote car diagnostics, vehicle health reports, and Turn-By-Turn Navigation to network access device services (Auto Connected Car, 2014). The communication between the car and the internet facilitates access to a variety of data sources. It can provide drivers with traditional real-time navigation, road information, and radio functions offered by the dashboard. In addition to more interesting advanced multimedia and infotainment services, it can reduce driver's driving stress and improved safety (Coppola & Morisio, 2016). Driving safety is the most important factor in the design of next-generation vehicles (Golestan et al., 2016). As the family of smartphones and connected devices continues to spread, they are integrated with on-board dashboards.

That allows data collected by internal car sensors to be combined with information collected from the network and the surrounding environment, providing drivers with more convenient and time-saving driving information. (Coppola & Morisio, 2016.) When necessary, the driver can be provided with relevant safety information in time, which helps the driver understand the driving situation and makes appropriate decisions based on the specific situation to avoid potential danger (Golestan et al., 2016). Most recent literature demonstrates that continuous connection to the internet and the existence of on-board instruments with internet-related services are essential elements of a connected car. Simultaneously, connectivity with smartphones is becoming increasingly important (Coppola & Morisio, 2016).

Overview of the connected car reveals the following characteristics:

- Internet access via built-in or user devices
- Having a set of modern car applications and dynamic context features can provide advanced infotainment functions to the user
- Vehicles can communicate with each other
- Vehicles can interact with smart devices in road infrastructure. (Coppola & Morisio, 2016.)

With the rapid rise of the IoT, referring to a global network of uniquely addressable interconnected objects based on standard communication protocols whose convergence point is the internet (Botta, De Donato, Persico, & Pescapé, 2016). IoT-related technologies pave the way for the automotive industry, and connected cars will play an essential role in roads and future city construction (Krasniqi & Hajrizi, 2016). Connected cars are quickly becoming a vital milestone for the design of intelligent transportation systems as a critical element in the design of smart and connected cities. The goal is to achieve ultra-efficient navigation and safer travel, improve road safety, reduce traffic accidents and improve people's quality of life. IoT brings everything together; sensors are generally integrated into the physical infrastructure. (Golestan et al., 2016.) According to forecasted by Statista, the number of global IoT connected devices in 2020 will be 30.73 billion (Priyadharshini, Ponmurugan, Nishanthi, & Elzalet, 2019). Gartner's statistic shows that there will be between 20.4 billion and 31 billion IoT devices online by 2020 (Vega, 2020). Furthermore, enterprise and automotive IoT endpoints will be 5.8 billion in use in 2020, a 21% increase from 2019 (Petrov, 2020). Billions of intelligent sensors have been embedded in the environmental equipment we live in, such as cars, streets, and surrounding buildings. These devices are expected to automatically discover their environment, connect and interact with the space around them and be able to send data streams for various targets. (Golestan et al., 2016.)

Figure 3 shows the architecture of the internet of cars, also called connected cars here. It consists of the client, connection and cloud layers. The vehicle is regarded as a mobile sensor platform. All sensors in the car are located on the client layer. The communication links of each of the vehicle-to-RSU (Road-side Units) (V2R), vehicle-to-infrastructure (V2I), vehicle-to-vehicle (V2V), vehicle-to-human (V2H), and vehicle-to-sensor (V2S) units can collect data from the surrounding environment. This data is further used to detect various situations of interest, i.e., driving environment and vehicle conditions. The connection layer handles different types of wireless communication on the internet of cars, like V2R, V2I, V2H and V2S to ensure connectedness and roaming with the existing network. These networks include the vehicular ad-hoc network (VANET), Universal Mobile Telecommunications System (UMTS) and Long Term Evolution (LTE). The client layer uses the intra-car, inter-car and vehicle network communication links provided by the connection layer to import VANET data. It then transmits the relevant

information to the entity of interest. The cloud layer offers internet access to mobile cars for cloud-based processes, in other words access to computing resources, content search, spectrum sharing, etc. Through this architecture, various sensors installed in the car can receive and generate a large amount of data and information from relevant sensor information sources and provide drivers with much data about the external environment, such as weather, road conditions, traffic, and social network flows. (Golestan et al., 2016.)

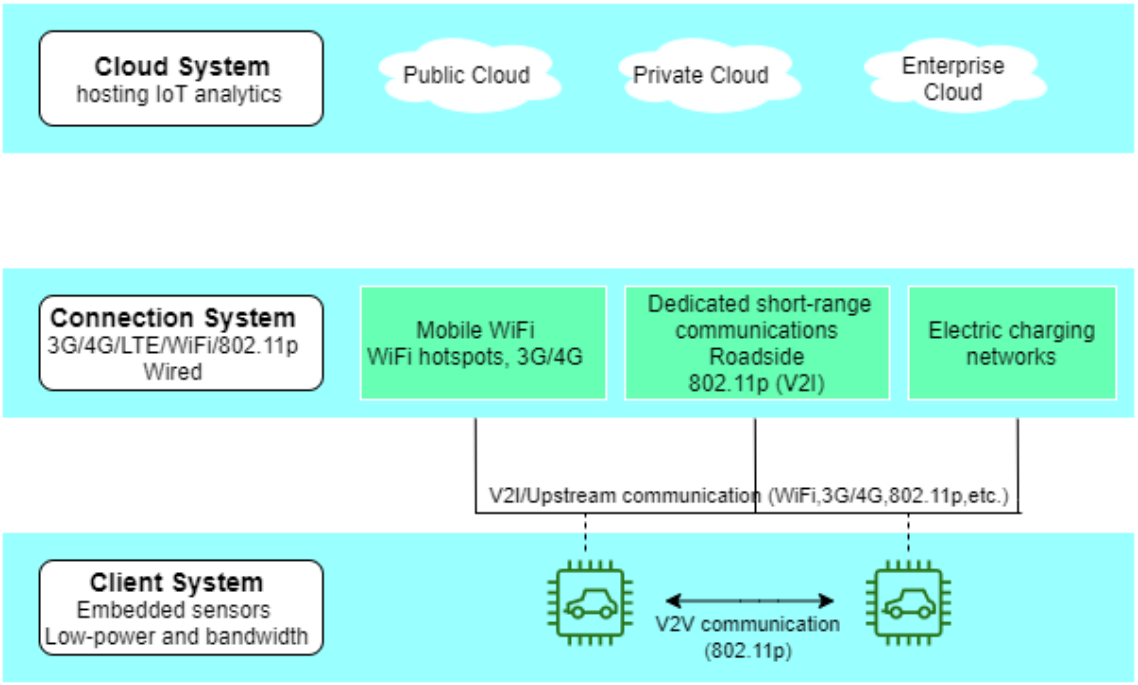


Figure 3. The three-layer client–connection–cloud architecture for the internet of cars (adapted from Golestan et al., 2016)

Over the next 30 years, continuous technological innovation will make car travel more convenient (Buehler, 2018). Connected and automated vehicles (CAVs), also known as connected and autonomous vehicles and driverless cars (Elliott, Keen, & Miao, 2019), will perform more and more driving tasks without a driver, relying on sensors and the connection technologies to communicate with other vehicles and infrastructure (Buehler, 2018). Figure 4 is the visualization chart used by the Society of Automotive Engineers (SAE) International in its J3016TM "Levels of Driving Automation" standard. The first deployed chart was in 2016, the latest version updated on January 7, 2019. Real-time documentation defines six driving automation levels from SAE level zero (no automation) to SAE level 5 (full vehicle autonomy), clearly illustrating the six driving levels' functions. SAE level 0-2 requires people to operate the leading car; SAE Level 3 is driven by humans only when some automatic parts are requested. Level 4 is conditional automated driving, and ultimate level 5 is full automation. This chart aims to clarify and simplify the J3016 "Levels of Driving Automation" standard for consumers. It is the most-cited reference for automated vehicle (AV) capabilities in the industry, guiding manufacturers and other entities to safely design, develop, test, and deploy highly automated vehicles (HAV). The US Department of Transportation (DoT) used this table for on-road motor vehicles. This document has become a de facto global standard adopted by stakeholders in autonomous vehicle technology. (Shuttleworth, 2019; Warrendale, 2018.) Ultimately, the goal of the automotive industry is to achieve CAVs fully, enabling individuals to engage in activities other than driving cars (Buehler, 2018). The automotive industry is on the verge of a revolution in the transition to the autonomous vehicle industry. The driving force behind it is the rapid development of IoT technology. IoT will change

the automotive industry. At the same time, the automotive industry will significantly promote the development of the IoT. (Krasniqi & Hajrizi, 2016.)

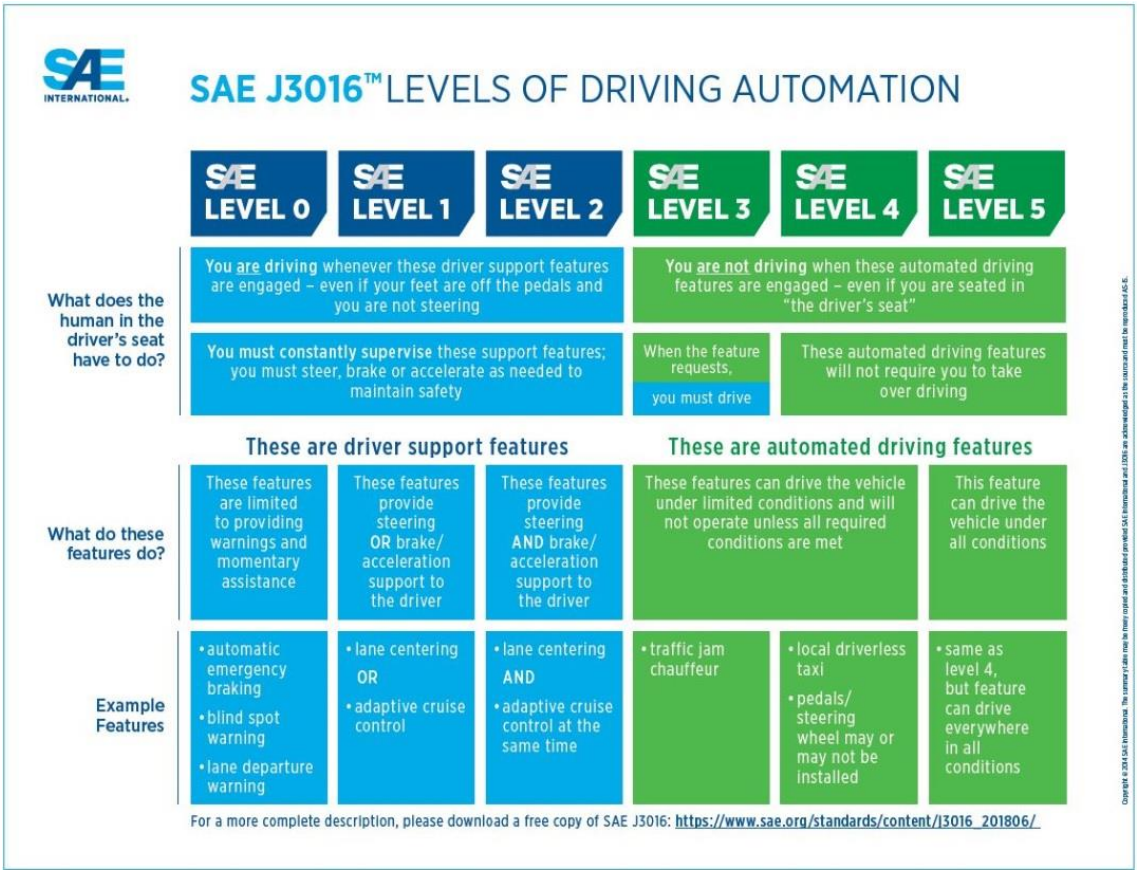


Figure 4. SAE J3016 levels of driving automation (source from Shuttleworth, 2019, January 7)

The applications required by the internet of cars can be broadly divided into three main categories: safety, convenience and comfort. Applications should be developed according to the requirements to achieve specific goals. Meanwhile, the challenges facing the internet of cars are highly focused on the following areas: routing and communication protocols, security and privacy, data distribution, simulation, information management and information fusion. Various sensors are installed in cars, and these sensors and sensor sources always generate and receive large amounts of data and information. Connected cars use big data and information pools to provide drivers with the information perception they need. Therefore, the available data, the diversity, and the massive scale of information are the main challenges facing an internet of cars. (Golestan et al., 2016.)

3.1.2 Car Navigation System

The development of car navigation systems is achieved with the continuous improvement of electronic equipment and the availability of location information sources such as GPS and digital maps. Generally, there are two types of car navigation systems. One is managed by a centralised system, which performs continuous two-way communication with vehicles that need navigation services. Information from the onboard vehicle sensors is transmitted to the navigation center which estimates the car's position and sends guidance commands back to the driver. Another type of autonomous navigation system does not require the navigation center's participation to process all the information and calculate the best route or necessary guidance commands. The autonomous navigation

system has become the standard in the automotive industry. (Obradovic, Lenz, & Schupfner, 2007.)

The car navigation system has three primary functions: positioning, routing, and navigation for guidance. Car positioning cannot separate from location information sources: GPS and digital maps. (Obradovic et al., 2007.) GPS and digital maps have become the main tools for vehicle positioning, providing vehicle location information and geometric previews of roads in use (Piao, Beecroft, & McDonald, 2010). GPS is the most frequently utilized and the earliest developed system from the Global Navigation Satellite System (GNSS) (Walker & Awange, 2018). GNSS is a standard generic term for satellite navigation systems that provide autonomous geospatial positioning with global coverage (Jackson, Polglaze, Dawson, King, & Peeling, 2018). It includes GPS, GLONASS (GLObal NAvigation Satellite System), BeiDou, Galileo, and other regional systems (Li, Zhang, Ren, Fritsche, Wickert, & Schuh, 2015). The development of GPS can be traced back to the 1960s. By 1973, the US military has decided to include GPS with timing and ranging in the navigation system, and it was fully operational in 1995. The overall goal was a tool for locating points on earth without using the ground's target. (Walker & Awange, 2018.) There were 24 satellites working on six orbital planes, and each orbital plane has four satellites on average, with an orbital inclination of 60 ° to surround the Earth (Zito, D'este, & Taylor, 1995). GPS provides real-time 3D positioning, navigation, and speed data (Walker & Awange, 2018). Anyone with a GPS receiver can access it for free (Ramani, Valarmathy, SuthanthiraVanitha, Selvaraju, Thiruppathi, & Thangam, 2013). Currently, 24 satellites of GPS and 24 satellites of GLONASS together make up a network of 48 satellites (Jackson et al., 2018). It can continuously transmit time and satellite identification information. GPS receivers are embedded in the vehicle devices to detect the signals and generate the positioning data and speed information as two separate outputs (Obradovic et al., 2007). The positioning data is one of the essential components of the car system (Lee, Tewolde, & Kwon, 2014), a necessary part of safe driving (Piao et al., 2010). Usually, the car positioning question aims to answer "Where am I?" If the system does not know where the car is, it isn't straightforward to determine what to do next (Rahiman & Zainal, 2013). Positioning is critical for many vehicle applications, usually using a GPS for positioning (Piao et al., 2010). The digital map contains information about the road network, including road attributes such as highways and one-way streets. It can also provide different services that passengers are interested in: the type and location of hotels and shopping malls, the cheapest gas station nearby and the available parking spots (Obradovic et al., 2007). Navigation information can also be displayed on a head-up display for easy viewing and avoid distracting drivers (Coppola & Morisio, 2016). Routes are calculated as the best route between points A and B relative to the selected standard. Once the best way and current location are known, the guidance algorithm will advise the driver to keep the vehicle on the chosen route. This is the navigation. (Obradovic et al., 2007.)

When faced with future development, the car navigation system is one of the critical parts of autonomous cars' core technology (Zhao, Liang, & Chen, 2018). The GPS-based autonomous car navigation system is a rapidly developing technology that uses real-time geographic data from multiple GPS satellites to calculate longitude, latitude, speed, and route every second to help drive cars. Researchers have developed a variety of technologies for navigating in various external environments. The GPS-based autonomous car navigation system has been widely used in land vehicle navigation applications. (Rahiman & Zainal, 2013.)

3.1.3 Vehicle Tracking System

The vehicle tracking system can inform the vehicle of the location and route (Kamel, 2015). It is an application widely used in many areas worldwide, such as transportation and logistics, shipping, fleet management, law enforcement and personnel monitoring. Tracking systems fall into two classifications, beacon-based systems and GPS-based systems. A beacon-based tracking system requires a large number of beacons to be connected to different areas to provide unique identification for the local area. The other system is a GPS-based tracking system. GPS receivers embedded in tracking devices receive radio signals from at least four satellites to generate the positioning data in real-time. This is the most full-grown and the most commonly used technology in the positioning and location tracking system. (Tang, Shi, & Lei, 2016.)

The GPS/GPRS (General Packet Radio Service) based vehicle tracking system is an essential system that integrates both GPS and GPRS technologies with hardware and software (Ramani et al., 2013). The vehicle tracking system works through the commonly used GPS technology to determine the location of a vehicle. The GPS module is integrated inside the vehicle tracking device to acquire geographic coordinates at fixed time intervals. It can track the car or fleet and obtain information about the vehicle's current location. GPRS technology is used to transmit and update vehicle location data between the in-vehicle unit and the tracking server. It sends the vehicle location and speed information to the server. (Kamel, 2015.) There are four components, hardware (i.e., the vehicle tracking device), software, the server and client (in-vehicle unit) interface. These four parts are indispensable: the vehicle tracking system will be completely ineffective if missing one of them.

Many researchers work on the vehicle tracking system to provide different solutions. Chadil, Russameesawang and Keeratiwintakorn (2008) proposed an open-source GPS tracking system called Goo-Tracking that using a client-server model: an embedded device with a GPS / GPRS module is used to identify location information and send it to the server regularly. The server is a personal computer with a web server program, used to receive location information, and then use the Google Earth software google map for location display.

With the development of IoT technology, car tracking technology now requires the deployment of some IoT components. The technical aspects of IoT car tracking systems include Radio Frequency Identification (RFID)/Sensors, GPS, Wireless Sensor Network (WSN), cloud computing, and application software. RFID performs non-contact, two-way data communication between the reader and the tag through wireless radio frequency to identify. (Thomas & Rad, 2017.) In a typical usage scenario of the IoT, the reader generates an appropriate signal to trigger the tag's transmission, checking whether the tag uniquely identifies an object. RFID tags are usually passive (no on-board power supply is required), but some tags use battery power. (Botta et al., 2016.) It can give a unique ID to any object for identification and communication. GPS is an integral part of the IoT technology paradigm. The tracked car receives data from sensors or RFID tags through the network. GPS sends data in bytes. The data contains location information of any vehicle. The GPS used in car tracking can easily navigate and monitor any car. (Thomas & Rad, 2017.) WSN usually consists of a potentially large number of small sensing nodes and communicates in a wireless multi-hop manner. Particular nodes or sinks are used to collect the results. (Botta et al., 2016.) It is an essential component in IoT and can provide an effective communication platform for different intelligent objects. In a network composed of different intelligent objects, and objects can be connected and exchange data simultaneously on a vast network segment. It supplies a network to collect analyzed,

processed and transmitted information between smart objects. (Thomas & Rad, 2017.) Cloud computing is described as a model that enables convenient, on-demand access to the network to configure configurable computing resources. It acts as a back-end for connecting significant data streams of things; it has unlimited capabilities in storage and processing capabilities and can be carried out quickly allocated and released. (Botta et al., 2016; Thomas & Rad, 2017.) For application software, its application interface is treated as the control medium to monitor and analyze the transition and calculate statistics (Thomas & Rad, 2017).

3.2 Background

The following section describes the background of this paper, Eclipse Kuksa and related APP4MC-Rover.

3.2.1 Eclipse Kuksa

Eclipse Kuksa also known as Eclipse Kuksa ecosystem or Kuksa is "**a software platform and ecosystem for Vehicle-2-X scenarios**" (Eclipse Kuksa, 2020). The idea of the project begins with the APPSTACLE goal (About Kuksa, 2018), ensuring that the automotive industry can take advantage of open source and get rid of its dependence on proprietary solutions (Appstacle, 2016). Figure 5 shows the structure of Eclipse Kuksa which adopts the development of cloud and vehicle infrastructure for V2X scenarios. It includes three parts: an in-vehicle platform, a cloud platform and an App IDE. (Structure of Kuksa, 2018; IoT Kuksa, 2020.)

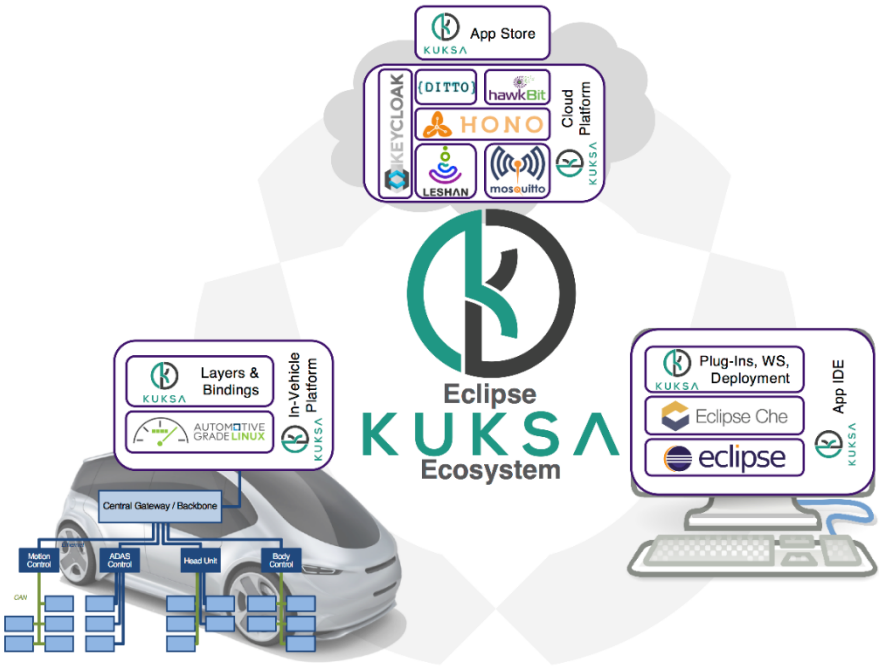


Figure 5. Eclipse Kuksa (source from Structure of Kuksa, 2018)

In-vehicle platform

The in-vehicle platform uses Automotive Grade Linux (AGL) as a base operating system. As a Linux Foundation collaborative project, AGL provides an open source platform for

the automotive industry. It brings together automotive manufacturers, suppliers, and technology companies to create a Linux-based open software platform to develop automotive software programs (Automotive Grade Linux, 2016). The in-vehicle platform works with AGL. AGL uses the Yocto/bitbake to build its components. The in-vehicle platform uses a wrapped method to add Eclipse Kuksa specific bitbake layer on the top of the original AGL to build Eclipse Kuksa adopted AGL. The in-vehicle component assures bidirectional communication with the outside world through the engine control unit (ECU). At the same time, it also contains a variety of technologies that can perform updates, upgrades, maintenance, diagnostics, as well as various data exchanges in a secure, error-free, authenticated, and verified manner. (Structure of Kuksa, 2018.) Besides AGL, the in-vehicle platform support with the different operating systems, such as Raspbian, Debian, Apertis, etc., which need a support Docker (IoT Kuksa, 2020).

Cloud platform

The cloud platform provides services that interact with the vehicles to manage them and provide interfaces with third-party services (Kuksa documentation, 2019). The cloud platform integrates multiple open source projects of Eclipse IoT, such as Eclipse Hono, Eclipse hawkBit, and Eclipse Ditto (Eclipse Kuksa, 2020). Eclipse Hono provides a remote service interface that connects many IoT devices to the backend and interacts with them in a unified way, regardless of the device communication protocol (Eclipse Hono, 2019). They help provide universal deployment based on Kubernetes to set up the automotive IoT cloud backend. Keycloak is used for authentication and authorization. The cloud platform also provides an app store that is user-centric and uses hawkBit as the base technology for automatic deployment of in-vehicle applications. (Eclipse Kuksa, 2020).

App IDE

The app IDE provides various API (Application Programming Interface)s to supply services for developers. Whether it is a vehicle application implemented by the Eclipse Kuksa app IDE based on Eclipse Che, or the Eclipse Kuksa app IDE based on VSCode, or provide built applications in the app store. These enable accessing existing communication interfaces for secure data transmission, storage, management, and authentication. Eclipse Kuksa also supports simplifying the deployment of new applications for cloud and vehicle components. The app IDE provides a simple and straightforward mechanism; configuration, building and deployment can be completed at the push of a button without further configuration or processing. (Kuksa documentation, 2019.) The app IDE supplies automation for the development of in-vehicle applications (IoT Kuksa, 2020).

Put together, the in-vehicle platform, the cloud platform and the app IDE form a complete toolset for the connected car field, providing various frameworks and technologies. Meanwhile, it can make use of new technologies like IoT and 5G connectivity. They allow people to combine vehicles, IoT, cloud and security technologies to make it easier to do development work for suppliers, OEMs, enterprises, and developers. (About Kuksa, 2018.)

3.2.2 APP4MC-Rover

APP4MC-Rover, also called Rover, is an open-source mobile robot described in Figure 6 (Rover intro, 2017). APP4MC (Eclipse APP4MC) is an open platform for engineering,

with embedded multi-core and multi-core software systems. The platform can create and manage complex toolchains, including simulation and verification. It has been further developed collaboratively with the Eclipse Kuksa project to demonstrate Eclipse APP4MC and the results of the Eclipse Kuksa research projects. APP4MC supports interoperability and scalability and unifies data exchange in cross-organization projects (Eclipse App4mc, 2020). Rover has applications designed for cloud communications, open-source tools, cluster computing and other complex research fields. Rover uses the Eclipse Paho MQTT (Message Queuing Telemetry Transport) client to connect to the cloud instance's message gateway to send telemetry data and receive driving commands. (Rover intro, 2017.) Eclipse Paho is an Eclipse IoT project that provides open-source MQTT and MQTT-SN (Sensor Network) in various programming languages, mainly for client-side implementation (Eclipse Paho, 2020). Rover is equipped with powerful sensors, motors and display units, such as OLED displays, to interact with the physical world. The Rover software is called Roverapp. It is designed to run on a Linux-based embedded single-board computer Raspberry Pi with a single executable file (Rover intro, 2017). Raspberry Pi is a Linux-based, small (credit card-sized), powerful, low-cost, single-board computer, launched in 2012. It operates in the same way as a standard PC that requires a keyboard, display unit and power supply. It has universal input/output, and the internet connection can be via Ethernet/LAN cable or WiFi connection. It is an ideal platform for connecting with many devices. (Maksimović, Vujović, Davidović, Milošević, & Perišić, 2014; Ibrahim, Elgamri, Babiker, & Mohamed, 2015.)

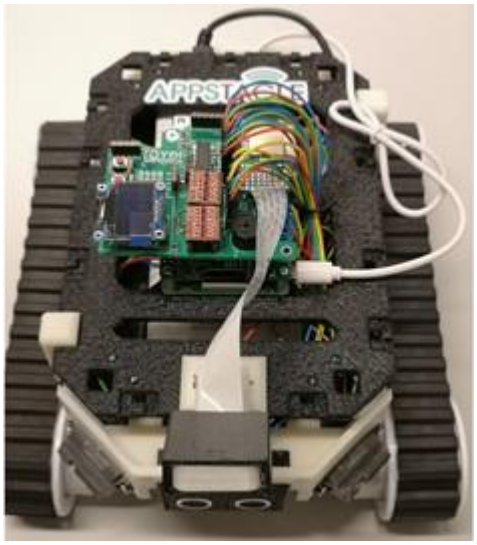


Figure 6. The Rover used at University of Oulu.

4. Research Implementation

This chapter describes how to use the DSRM Process to design and implement a vehicle navigation application for Eclipse Kuksa, including requirements, design, development, demonstration, and evaluation phases covering the entire development process. This thesis worked on the vehicle navigation application included two sub-applications; one was designed and implemented on the in-vehicle platform; the other was on the cloud platform. The application of the in-vehicle part was the client application running on the Raspberry Pi. MQTT protocol was used to communicate between the in-vehicle part and the cloud.

4.1 Requirements

This research aimed to investigate a GPS-based vehicle navigation application on the in-vehicle and cloud platform of Eclipse Kuksa. Therefore, Eclipse Kuksa's in-vehicle and cloud platforms had conducted their research. The research's primary research question was how to develop a GPS-based vehicle navigation application using the Eclipse Kuksa software platform. The solution to the problem was to meet the GPS-based vehicle navigation application requirements on the in-vehicle and cloud platform of Eclipse Kuksa. The relevant vehicle navigation applications' requirements were obtained from the corresponding literature review, and the specific needs were from the Eclipse Kuksa development environment. The concluded requirements of the GPS-based vehicle navigation application for Eclipse Kuksa were described in Table 1.

From Table 1, Firstly, this GPS-based vehicle navigation application's **use case** was to display the car's route and save the route information.

The information from the literature review that positioning is the primary function of the car navigation system. The car's position information comes from GPS. The GPS module is embedded into the vehicle (Maurya, Singh, & Jain, 2012). The GPS module has a GPS receiver with an antenna to obtain geographic coordinates at a fixed time interval, the vehicle's current location information (Lee, Tewolde, & Kwon, 2014). So **R1** was generated, that the application should get the vehicle's current location information regularly. The geographical coordinates are written as (latitude, longitude) (Maurya, Singh, & Jain, 2012); the application could get it as a function too, which was defined as **R2**. The vehicle navigation application needs to display the car's route. The vehicle tracking system can inform the vehicle of the location and route. The system adopts a client-server model. The GPS module embedded in the car (client) is used to identify location information and send it to the server regularly; the server is used to receive location information and then use the map tool to display the car's location. The Eclipse Kuksa is the client (in-vehicle platform) and the server (cloud platform). R1, R2 was ready at the in-vehicle platform; the only need to send the vehicle location information to the cloud was **R3**. Next, move to the cloud platform. Referenced from Kamel (2015), the application got the vehicle location information at the cloud platform and then displayed graphically and saved the route information in a database. The database is used to store and manage the received vehicle location information. Whenever the user needs the location of the vehicle, it can be obtained from the database. **R4, R5,** and **R6** had generated accordingly. R1, R2, R3, R4, R5, and R6 were the GPS-based vehicle navigation application's functional requirements.

Table 1. The requirements of the GPS-based vehicle navigation application for Eclipse Kuksa.

Requirement ID	Requirement description
Functional requirements	
In-vehicle	
R1	The application should be able to capture the position of the vehicle (Maurya, Singh, & Jain, 2012; Lee, Tewolde, & Kwon, 2014).
R2	The vehicle's position is a function of the GPS coordinates (Maurya, Singh, & Jain, 2012).
R3	The application should transmit the position of the vehicle to the cloud (Kamel, 2015).
Cloud	
R4	The application should be able to get the vehicle's in-vehicle transmitting position (Kamel, 2015).
R5	The application should store the vehicle position and time information in a database (Kamel, 2015).
R6	The application should provide users to monitor the vehicle's position continuously (Kamel, 2015).
Non-Functional requirements	
R7	The communication between the in-vehicle application and the cloud application is through the MQTT protocol (Rover intro, 2017).
R8	The communication between the in-vehicle platform and the cloud platform is through the Eclipse Hono gateway (Kuksa documentation, 2019; Rover intro, 2017).
R9	The in-vehicle application should be compatible with Kuksa AGL (Automotive Grade Linux) (Structure of Kuksa, 2018).
R10	Both the in-vehicle application and the cloud application should execute correctly, without failure or faults (Thomas and Rad, 2017).

Meanwhile, combined with the Eclipse Kuksa development environment's characteristics such as AGL open operating system, Hono gateway, and MQTT protocol used to communicate between the client and the cloud, non-functional requirements **R7, R8,** and **R9** were determined. The car navigation application is part of many applications of the IoT. According to Thomas and Rad (2017), The IoT system needs reliability evaluation to increase quality assurance. Reliability Evaluation requires that the system operate correctly without failure, failure, or error; **R10** was required. R7, R8, R9, R10 were the application's Non-functional requirements.

4.2 Designing

According to the requirement and the Eclipse Kuksa development environment, the GPS-based vehicle navigation application's system architecture in the Eclipse Kuksa environment in real life in Figure 7. There was a GPS Receiver model in the vehicle to continuously receive information from GPS satellites and then calculates the vehicle's geographic location. To send this receiving vehicle location information to the cloud through the internet connection: Cellular network 3G, 4G, LTE, even 5G. Graphically displayed the location information on the cloud device, such as a mobile phone, computer,

etc., also saved the location information to the database, and the 3rd party product can pick up the data for further development.

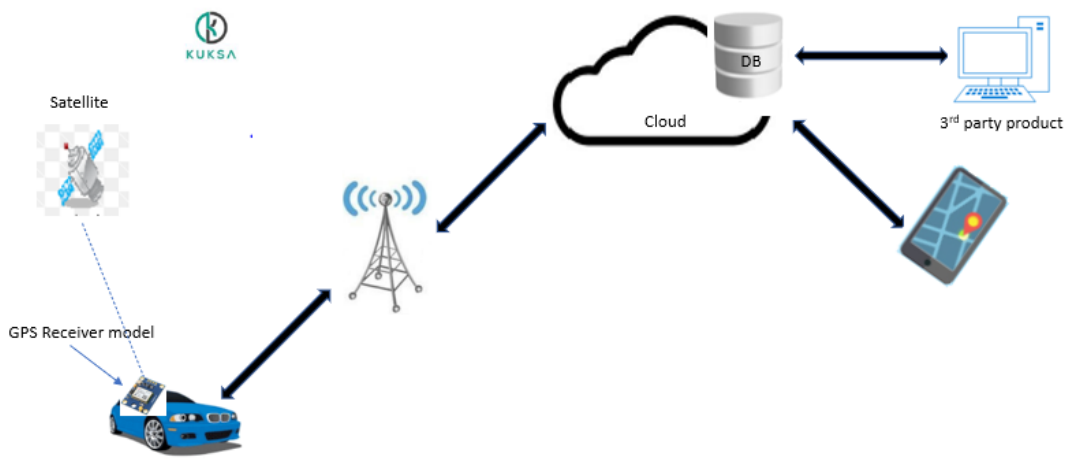


Figure 7. System architecture of the GPS-based vehicle navigation application in real life.

As illustrated in Figure 8, the internet connection used wireless network Wi-Fi in the laboratory. The implementation applied the Rover to develop the vehicle navigation application. Planned Eclipse Kuksa to have a GPS receiver model on the Rover. But there was no GPS receiver model on the Rover, so the implementation utilized the mock GPS to demonstrate the GPS receiver model to produce the GPS data in-vehicle platform; this means obtained the GPS data from a recorded GPS file. Used Rover features to communicate between the in-vehicle and the cloud platform through the MQTT protocol. The initial design was to send vehicle location information to the cloud through the Hono gateway. However, during the research development process, with the end of the Eclipse Kuksa project, the Hono gateway was shut down; therefore, the spring boot was designed to be adopted to receive the MQTT client's transmission. When the Hono gateway is available, it can be added and tested to receive the MQTT client's message through the Hono gateway and then to the spring boot web service. At the cloud, storing the received data in the database of InfluxDB. Also, the implementation utilized google maps to display the location information of the vehicle graphically.

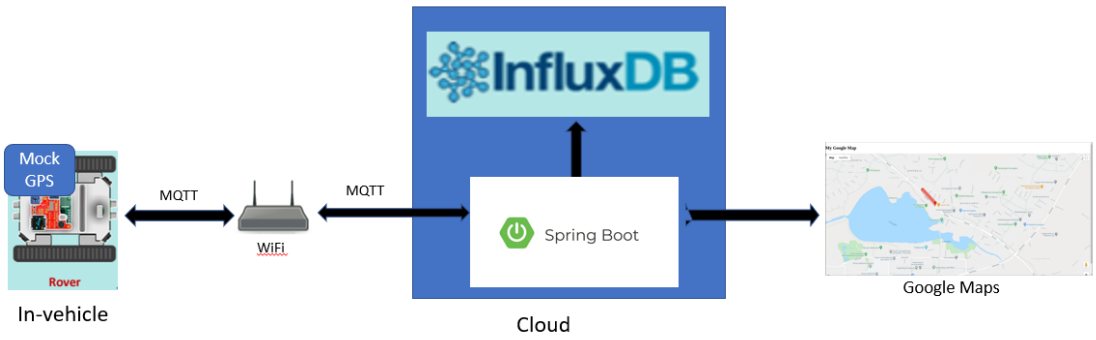


Figure 8. System architecture of the GPS-based vehicle navigation application in laboratory.

Figure 9 shows the MQTT architecture. The in-vehicle application on the client-side sent the vehicle location information through MQTT publish payload to the server side's

MQTT broker; the cloud application on the server-side got the vehicle location information through the MQTT subscribe topic.

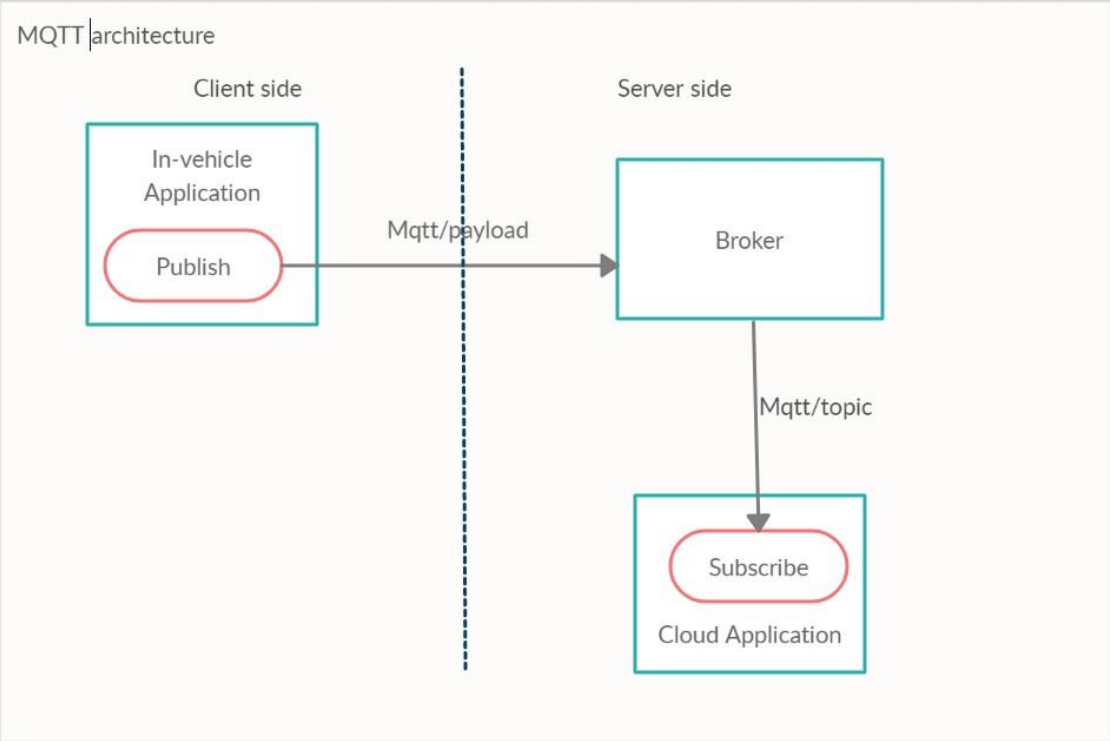


Figure 9. MQTT architecture.

4.3 Development

The development included two separate applications, which were programmed separately; one was an in-vehicle application running on the in-vehicle platform; the in-vehicle application was also referring as the client application in this study. The other was a cloud application running on the cloud platform. The communication between the in-vehicle and the cloud platform was through the MQTT protocol.

The source codes of the in-vehicle application and the cloud application were available from the author's personal GitHub account. The name of the GitHub project was NavigationApp, and the link was https://github.com/Hong0802/NavigationApp.

4.3.1 In-vehicle application

The in-vehicle application was programmed with Java and run from the in-vehicle platform. The in-vehicle application features included reading the vehicle's position data from a recorded GPS file, setting up MQTT protocol, and publishing data via MQTT protocol.

Read the position data of the vehicle from a recorded GPS file

A recorded GPS, a ready-made GPS data file, was used because the GPS receiver model was not available on the Rover. The GPS data was captured from the mobile device and saved with the real data. Figure 10 was the part of the actual GPS data in the recorded GPS file *gps-data.txt*:

{"x":25.47841961401770, "y":65.07280578236698},
{"x":25.47874336979583, "y":65.07271102376659},
{"x":25.47889625446883, "y":65.07265795880306},
{"x":25.47904913914184, "y":65.07260489373380},
{"x":25.47920202381484, "y":65.07255182855877},
{"x":25.47935490848785, "y":65.07249876327805},
{"x":25.47950779316086, "y":65.07244190750274},
{"x":25.47966067783387, "y":65.07238505160605},
{"x":25.47981356250687, "y":65.07232819558794},

Figure 10. The part of the actual GPS data in the recorded GPS file *gps-data.txt*.

The Java-implementation in the *ClientApplication.java* gets the GPS data from the recorded file line by line every second.

```
Scanner scanner = new Scanner(new File("src/main/resources/gps-
data.txt"));
while (scanner.hasNextLine()) {
        String line = scanner.nextLine();
        if(line.contains("x")) {
                // process the line

                System.out.println(line);
                TimeUnit.SECONDS.sleep(1);
        }
}
```

Setup MQTT connection

MQTT protocol was used to communicate between the client app and the cloud app. Setting up an MQTT connection in the client app was described below.

The following was the configuration of the MQTT connection in the property file *application.properties* of the client app.

```
mqtt.automaticReconnect=true
mqtt.cleanSession=true
mqtt.connectionTimeout=10
mqtt.clientId=spring-client
mqtt.hostname=192.168.43.217
mqtt.port=1883
```

The codes of setting up MQTT connection in *MqttConfiguration.java* as the following.

```
public IMqttClient mqttClient(@Value("${mqtt.clientId}") String
clientId,
                              @Value("${mqtt.hostname}") String
hostname, @Value("${mqtt.port}") int port) throws MqttException {
        IMqttClient mqttClient = new MqttClient("tcp://" + hostname +
":" + port, clientId);
        mqttClient.connect(mqttConnectOptions());
        return mqttClient;
}
```

Publish data via MQTT protocol

The Java-implementation in the *ClientApplication.java* publishes the data through the MessageService of MQTT after getting the GPS data from the recorded file line by line every second.

```
final String topic = "gps/data";
messagingService.publish(topic, line, 0, true);
```

The codes in the *MessagingService.java* as the following.

```
public void publish(final String topic, final String payload, int qos,
boolean retained)
        throws MqttPersistenceException, MqttException {
    MqttMessage mqttMessage = new MqttMessage();
    mqttMessage.setPayload(payload.getBytes());
    mqttMessage.setQos(qos);
    mqttMessage.setRetained(retained);

    mqttClient.publish(topic, payload.getBytes(), qos, retained);
    }
```

4.3.2 Cloud application

The cloud application was programmed with Java and run from the cloud platform. The cloud application features included setting up MQTT protocol, subscribe data via MQTT protocol, saving the data to the InfluxDB database, creating a webpage to show the data, and drawing the route via Google Maps.

Setup MQTT connection

The communication between the client and the cloud app was through the MQTT protocol. Described the setting up of an MQTT connection in the cloud app was below.

The following was the configuration of the MQTT connection in the property file *application.properties* of the cloud app.

```
mqtt.automaticReconnect=true
mqtt.cleanSession=true
mqtt.connectionTimeout=10
mqtt.clientId=spring-server
mqtt.hostname=127.0.0.1
mqtt.port=1883
```

The codes of setting up the MQTT connection in *MqttConfiguration.java* as the following.

```
public IMqttClient mqttClient(@Value("${mqtt.clientId}") String
clientId,
                              @Value("${mqtt.hostname}") String
hostname, @Value("${mqtt.port}") int port) throws MqttException {
        IMqttClient mqttClient = new MqttClient("tcp://" + hostname +
":" + port, clientId);
        mqttClient.connect(mqttConnectOptions());
        return mqttClient;
}
```

Subscribe data via MQTT protocol

The following codes in the *DemoApplication.java* subscribe data through the MQTT protocol.

```
final String topic = "gps/data";
messagingService.subscribe(topic);
```

Save the data to InfluxDB database

The following codes in the *MessagingService.java* save the GPS data to the InfluxDB database. Firstly, connect to the database, and then save the GPS data to the table name is *gpsdata*, and the database name is *mydatabase*.

```
InfluxDB influxDB = InfluxDBFactory.connect("http://127.0.0.1:8086");

Point.Builder builder = Point.measurement("gpsdata");
builder.time(System.currentTimeMillis(), TimeUnit.MICROSECONDS);
builder.addField("lat",lat);
builder.addField("lng",lng);
builder.tag("device-id","rover");
Point point = builder.build();
influxDB.setDatabase("mydatabase").write(point);
```

Create a webpage to show the data

The following codes in the *GreetingController.java* assign latitude and longitude values to the latitude and longitude variables on the web page named "*location"*.

```
@GetMapping("/location")
public String locationSubmit(@ModelAttribute Greeting greeting, Model
model) throws JsonProcessingException {

        model.addAttribute("lat", messagingService.getLatitude());
        model.addAttribute("lng", messagingService.getLongitude());
        return "location";
}
```

Draw the route via Google Maps

The next part of Javascript code was a snippet from a *location.html* file used to draw the route through google maps with a maker.

```
var map = new google.maps.Map(document.getElementById('map'),
options);
var latitude = Number([[${lat}]]);
var longitude = Number([[${lng}]]);
var marker = new google.maps.Marker({
position:{lat:latitude,lng:longitude}, map:map,});
```

4.4 Demonstration

The demonstration was running the implementation of in-vehicle and cloud applications. As described in Figure 8, it needed a Raspberry Pi, which was on the Rover, a computer or laptop with Linux OS; also, a WLAN network required to demonstrate. The demonstration was to start the MQTT broker, start the cloud application, run the client application, and check the GPS data from the InfluxDB and the web browser route. There was a video to record the demonstration at the end of this section.

Start the MQTT broker

As illustrated in Figure 11, at the cloud side, typed the command "*mosquitto*" to start the MQTT broker, which was the mosquitto MQTT server had started, and its port was 1883.

```
hong@hong-ThinkPad-T510:~$ mosquitto
1599336382: mosquitto version 1.4.15 (build date Tue, 18 Jun 2019 11:42:22 -0300) starting
1599336382: Using default config.
1599336382: Opening ipv4 listen socket on port 1883.
1599336382: Opening ipv6 listen socket on port 1883.
```

Figure 11. Start the MQTT broker at the cloud side.

Start the cloud application

Opening the cloud application at the cloud side, started the cloud application by clicking the "Run" button, and displayed on the bottom's the running result, which was in Figure 12.

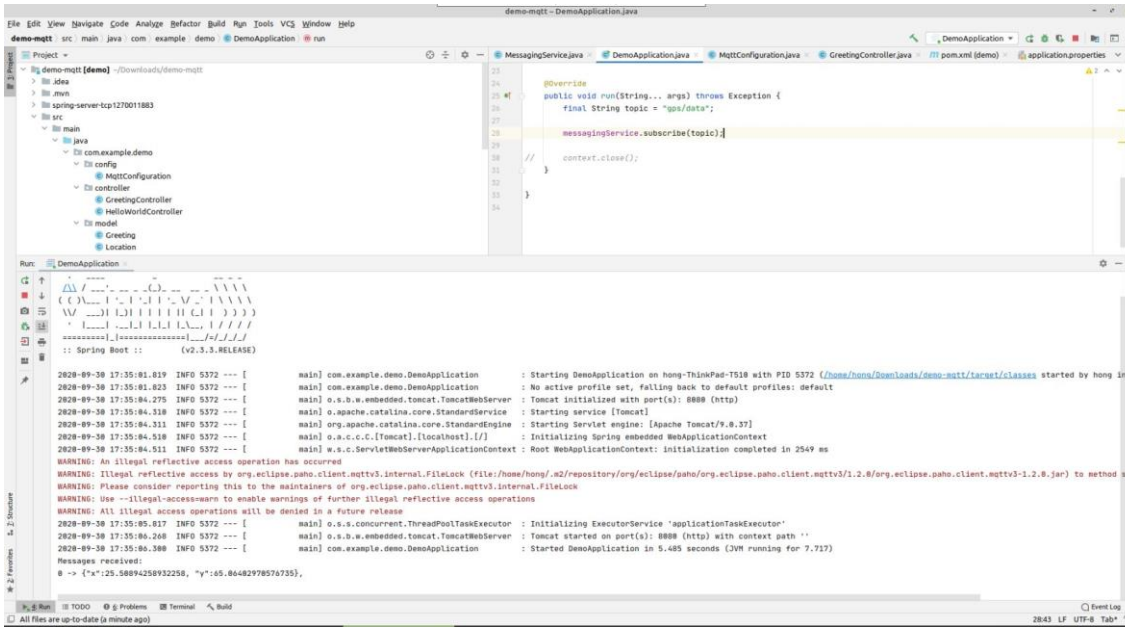


Figure 12. Start the cloud application from the cloud platform

Start the client application

The client application was opening and running in the Raspberry Pi, typing the command "*mvn spring-boot: run*" to start the client application. The running result as in Figure 13. Displayed the GPS data was one by one at the bottom of Figure 13.

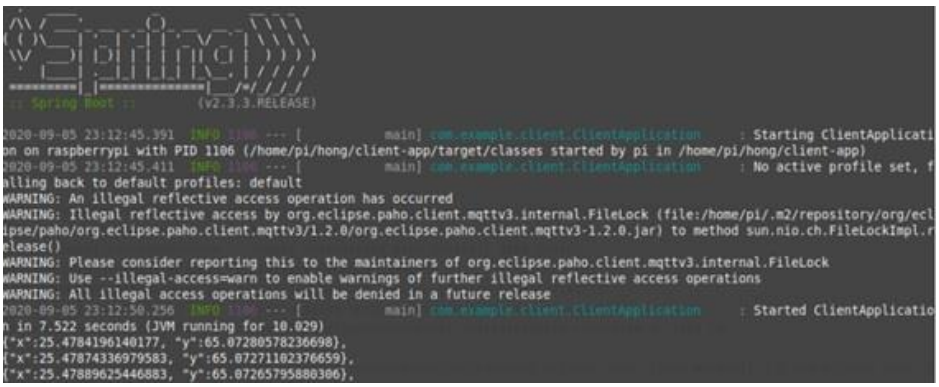


Figure 13. Start the client application from the Raspberry Pi of the in-vehicle platform.

Check the GPS data from the InfluxDB

Back to Figure 12. The GPS data of the client application sent was displayed one by one in the cloud application running result. Connected into the InfluxDB, checked the GPS data from the *gpsdata* table by typing the command "*select * from gpsdata*", saved the GPS data of the client application sent was in the *gpsdata* table in Figure 14.

```
> use mydatabase
Using database mydatabase
> select * from gpsdata
name: gpsdata
time                device-id       lat                 lng
----                ---------       ---                 ---
1599426145377000    rover           65.07126686096795   25.48240360873191
1599426178148000    rover           65.07126686096795   25.48240360873191
1599426186087000    rover           65.07280578236698   25.4784196140177
1599426187122000    rover           65.07271102376659   25.47874336979583
1599426188126000    rover           65.07265795880306   25.47889625446883
1599426189127000    rover           65.0726048937338    25.47904913914184
1599426190128000    rover           65.07255182855877   25.47920202381484
1599426191133000    rover           65.07249876327805   25.47935490848785
1599426192133000    rover           65.07244190750274   25.47950779316086
1599426193133000    rover           65.07238505160605   25.47966067783387
1599426194133000    rover           65.07232819558794   25.47981356250687
1599426195136000    rover           65.07227133944849   25.47995745396382
1599426196135000    rover           65.07221448318761   25.48011033863682
1599426197137000    rover           65.07215762680535   25.48025423009377
1599426198140000    rover           65.0721007703017    25.48039812155072
1599426199141000    rover           65.07204391367665   25.48054201300767
1599426200142000    rover           65.07198705693021   25.48068590446462
1599426201142000    rover           65.07192640960021   25.48082979592156
```

Figure 14. The saved GPS data in the gpsdata table of InfluxDB.

Open the web browser to see the route

Opening a web browser and typing "*localhost:8080/location*", displayed the route formed by GPS data was on the webpage in Figure 15.

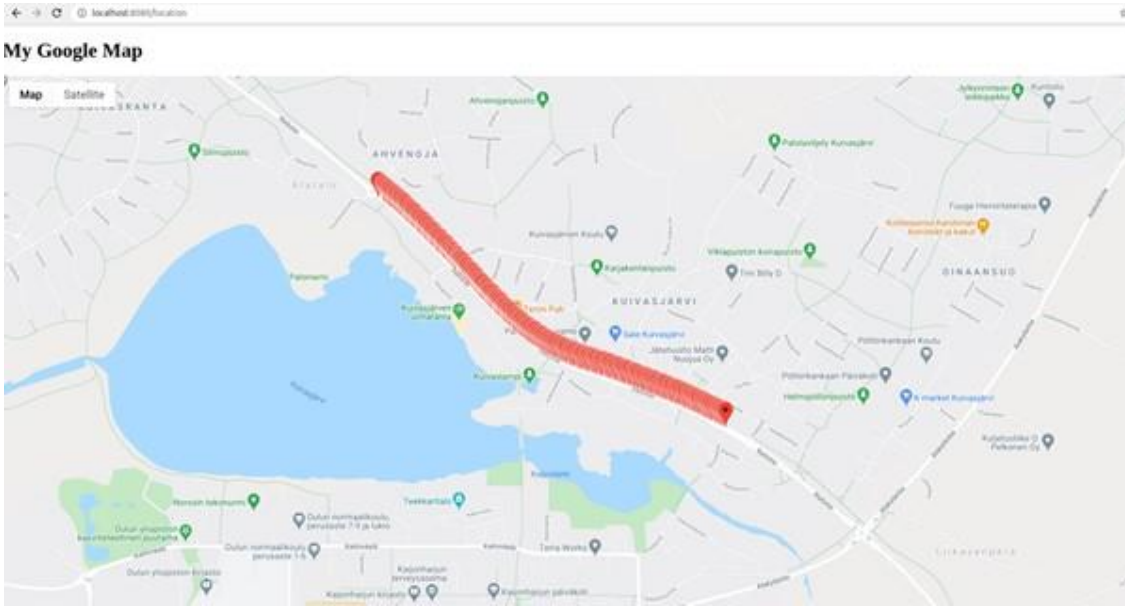


Figure 15. The route formed by GPS data on the google map.

The link of the video record of this demonstration was at https://drive.google.com/drive/folders/1JVfrHpFCZGhEiYVcYPoUlKC_ysoFL_Zs?usp=sharing

4.5 Evaluation

This part was about the evaluation of in-vehicle and cloud application development. Designed the test cases based on 4.1 requirements, executed the test cases, and check the execution results to review whether they had achieved the requirements. The evaluation was divided into three levels: "achieved, partially achieved, not achieved". Simultaneously giving evaluation notes and formed an evaluation report finally.

Test Case for R1: The application should be able to capture the position of the vehicle.
Test Case ID: TC1
Description: Start the client application by typing the command "*mvn spring-boot: run*"
Expected Result: The client application started with the following messages displayed on the screen.
"{"x":25.4784196140177, "y":65.07280578236698},
{"x":25.47874336979583, "y":65.07271102376659},
{"x":25.47889625446883, "y":65.07265795880306},
{"x":25.47904913914184, "y":65.0726048937338},
{"x":25.47920202381484, "y":65.07255182855877},
{"x":25.47935490848785, "y":65.07249876327805},
{"x":25.47950779316086, "y":65.07244190750274},
…"
Actual Result: The client application started with the messages displayed correctly.
Passed/Failed: Passed
Test Comment:

Test Case for R2: The vehicle's position is a function of the GPS coordinates.
Test Case ID: TC2
Description: Executing TS1 and the messages displayed on the screen as the function of the GPS coordinates {x, y} which is fitting with the geographical coordinates are written as (latitude, longitude)

Expected Result: The messages are displayed on the screen should be according to the following format.
"{"x":25.4784196140177, "y":65.07280578236698},
{"x":25.47874336979583, "y":65.07271102376659},
{"x":25.47889625446883, "y":65.07265795880306},
{"x":25.47904913914184, "y":65.0726048937338},
{"x":25.47920202381484, "y":65.07255182855877},
{"x":25.47935490848785, "y":65.07249876327805},
{"x":25.47950779316086, "y":65.07244190750274},
…"
Actual Result: The messages are displayed according to the expected format.
Passed/Failed: Passed
Test Comment:

Test Case for R3: The application should transmit the position of the vehicle to the cloud.
Test Case ID: TC3
Description: Firstly, start the MQTT broker by typing the command "mosquito"; secondly, start the cloud application by clicking the "Run" button; lastly, Start the client application by typing the command "mvn spring-boot: run"
Expected Result: In the screen of the cloud application running, the following messages are displayed, (the in-vehicle application sends those)
"Message received:
0 -> {"x":25.4784196140177, "y":65.07280578236698},
0 -> {"x":25.47874336979583, "y":65.07271102376659},
0 -> {"x":25.47889625446883, "y":65.07265795880306},
0 -> {"x":25.47904913914184, "y":65.0726048937338},
0 -> {"x":25.47920202381484, "y":65.07255182855877},
0 -> {"x":25.47935490848785, "y":65.07249876327805},
0 -> {"x":25.47950779316086, "y":65.07244190750274},
…"
Actual Result: The messages are displayed on the cloud application running's screen according to the expected.
Passed/Failed: Passed
Test Comment:

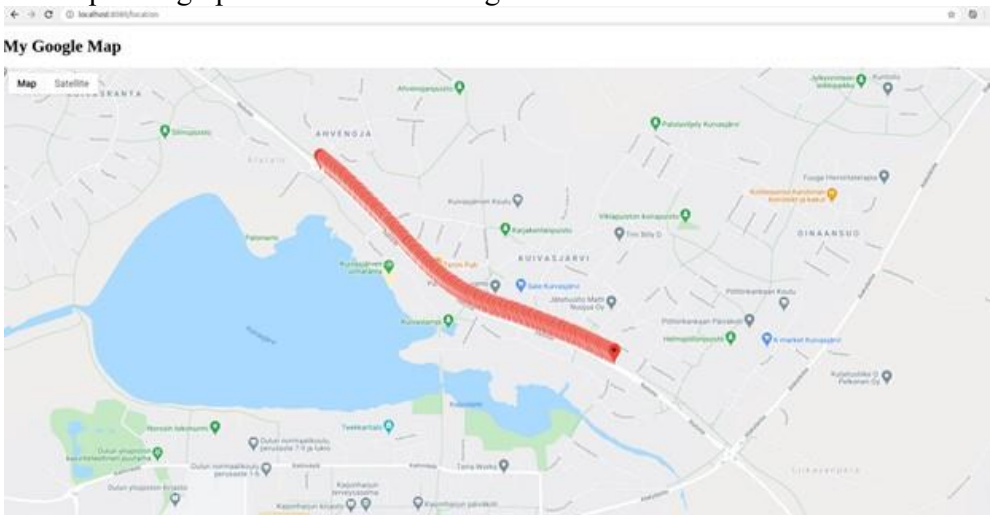
Test Case for R4: The application should be able to get the vehicle's in-vehicle transmitting position.
Test Case ID: TC4
Description: Executing TC3
Expected Result: On the screen of the cloud application running, the following messages are displayed.
"Message received:
0 -> {"x":25.4784196140177, "y":65.07280578236698},
0 -> {"x":25.47874336979583, "y":65.07271102376659},
0 -> {"x":25.47889625446883, "y":65.07265795880306},
0 -> {"x":25.47904913914184, "y":65.0726048937338},
0 -> {"x":25.47920202381484, "y":65.07255182855877},
0 -> {"x":25.47935490848785, "y":65.07249876327805},
0 -> {"x":25.47950779316086, "y":65.07244190750274},
…"
Actual Result: The messages are displayed on the cloud application running's screen according to the expected.
Passed/Failed: Passed
Test Comment:

Test Case for R5: The application should store the vehicle position and time information in a database.
Test Case ID: TC5
Description: Firstly, start the MQTT broker by typing the command "mosquito"; secondly, start the cloud application by clicking the "Run" button; thirdly, Start the client application by typing the command "mvn spring-boot: run" and lastly connect into the influxdb by typing "use mydatabase and check the GPS data from the gpsdata table by typing the command "select * from gpsdata"
Expected Result: There are following messages are displayed.

"time device-id lng lat
-------- ---------- --------- -----
15994261860887000 rover 65.07280578236698 25.4784196140177
15994261867122000 rover 65.07271102376659 25.47874336979583
15994261867126000 rover 65.07265795880306 25.47889625446883
…
"

Actual Result: The messages are displayed according to the expected.
Passed/Failed: Passed
Test Comment:

Test Case for R6: The application should provide users to monitor the vehicle's position continuously.
Test Case ID: TC6
Description: Firstly, start the MQTT broker by typing the command "mosquito"; secondly, start the cloud application by clicking the "Run" button; thirdly, Start the client application by typing the command "mvn spring-boot: run" and lastly, open a web browser, and type "*localhost:8080/location*" the route formed by GPS data is displayed on the webpage.
Expected Result: The vehicle's route is displayed on google maps continuously following the in-vehicle and cloud application running.
The expected graph is like the following:



Actual Result: The vehicle's route is displayed according to the expected.
Passed/Failed: Passed
Test Comment:

Test Case for R7: The communication between the in-vehicle application and the cloud application is through the MQTT protocol.
Test Case ID: TC7
Description 1: Firstly, start the cloud application by clicking the "Run" button; and then start the client application by typing the command "mvn spring-boot: run"

Expected Result 1: There is no message displayed on the cloud application's running screen.
Actual Result 1: No message displayed according to the expected.
Passed/Failed: Passed
Description 2: Firstly, start the MQTT broker by typing the command "mosquito"; secondly, start the cloud application by clicking the "Run" button; lastly, Start the client application by typing the command "mvn spring-boot: run"
Expected Result 2: On the screen of the cloud application running, the following messages are displayed.
"Message received:
0 -> {"x":25.4784196140177, "y":65.07280578236698},
0 -> {"x":25.47874336979583, "y":65.07271102376659},
0 -> {"x":25.47889625446883, "y":65.07265795880306},
0 -> {"x":25.47904913914184, "y":65.0726048937338},
0 -> {"x":25.47920202381484, "y":65.07255182855877},
0 -> {"x":25.47935490848785, "y":65.07249876327805},
0 -> {"x":25.47950779316086, "y":65.07244190750274},
…"
Actual Result 2: The messages are displayed on the cloud application running's screen according to the expected.
Passed/Failed: Passed
Test Comment: In description 1, without start MQTT broker and run the in-vehicle and cloud application, the result is no message; In description 2, firstly start MQTT broker and then run the in-vehicle and cloud application, the result is the messages are displayed. These proved the communication between the in-vehicle application and the cloud application is through the MQTT protocol.

Test Case for R8: The communication between the in-vehicle platform and the cloud platform is through the Eclipse Hono gateway.
Test Case ID: TC8
Description: Under production
Expected Result:
Actual Result:
Passed/Failed: Not execute
Test Comment: The test case design is not available due to the APPSTACLE project ended in December 2019, and this research could not anymore use that server. Therefore it couldn't use Eclipse Hono gateway.

Test Case for R9: The in-vehicle application should be compatible with Kuksa AGL (Automotive Grade Linux).
Test Case ID: TC9
Description: Under production
Expected Result:
Actual Result:
Passed/Failed: Not execute
Test Comment: The test case design is not available due to the in-vehicle application is not integrated with Kuksa AGL.

Test Case for R10: Both the in-vehicle application and the cloud application should execute correctly, without failure or faults.
Test Case ID: TC10
Description 1: start the MQTT broker by typing the command "mosquito" and then start the cloud application by clicking the "Run" button
Expected Result 1: The cloud application is running without any faults.
Actual Result 1: The cloud application is running according to the expected.
Passed/Failed: Passed

Description 2: Start the client application by typing the command “mvn spring-boot: run”
Expected Result 2: The client application started without any faults. On the screen of the cloud application running, the following messages are displayed.
“Message received:
0 -> {"x":25.4784196140177, "y":65.07280578236698},
0 -> {"x":25.47874336979583, "y":65.07271102376659},
0 -> {"x":25.47889625446883, "y":65.07265795880306},
0 -> {"x":25.47904913914184, "y":65.0726048937338},
0 -> {"x":25.47920202381484, "y":65.07255182855877},
0 -> {"x":25.47935490848785, "y":65.07249876327805},
0 -> {"x":25.47950779316086, "y":65.07244190750274},
…”
Actual Result 2: The client application is running, and the messages are displayed on the cloud application running’s screen according to the expected.
Passed/Failed: Passed
Test Comment: In description 1, the cloud application is running without faults; In description 2, The client application started without faults; and the messages are displayed on the cloud application running’s screen according to the expected, this proved both the in-vehicle application and the cloud application executed correctly without failure.

The evaluation is based on the testing result in Table 2.

Table 2. The evaluation report of the GPS-based vehicle navigation application for Eclipse Kuksa.

Requirement ID	Test Case ID	Status	Evaluation
R1	TC1	Achieved	Test case executed and passed.
R2	TC2	Achieved	Test case executed and passed.
R3	TC3	Achieved	Test case executed and passed.
R4	TC4	Achieved	Test case executed and passed.
R5	TC5	Achieved	Test case executed and passed.
R6	TC6	Achieved	Test case executed and passed.
R7	TC7	Achieved	Test case executed and passed.
R8	TC8	Not Achieved	Test case not executed; therefore, it does not achieve R8.
R9	TC9	Not Achieved	Test case not executed; therefore, it does not achieve R9.
R10	TC10	Achieved	Test case executed and passed.

5. Discussion

This research aims to investigate GPS-based vehicle navigation applications on the vehicle and cloud platforms of Eclipse Kuksa, determine how to develop a GPS-based vehicle navigation application using the Eclipse Kuksa software platform and discuss the benefits and challenges of Eclipse Kuksa software platform for developing vehicle applications. The research problems had been solved using the DSR methodology. The GPS-based vehicle navigation application development had been completed under the DSRM Process and evaluated according to the requirements identified in section 4.1. The evaluation results showed that the implementation met most of the requirements. The two research questions established at the beginning of the thesis research are answered and discussed below.

The DSR was carried out to answer the research questions raised in section 2.1. From the demonstration can see that the design has been successfully implemented using the Eclipse Kuksa software platform. This implementation process has answered *RQ1: How to develop a GPS-based vehicle navigation application using the Eclipse Kuksa software platform?* Following the steps described in the DSRM process, the author collected and determined the requirements, and designed according to them. Then implemented the design, and the executed application results verified that the requirements have been met. It also showed that Rover simplifies the research of vehicle application development in the laboratory.

The answer to *RQ2: What are the advantages and challenges of developing a vehicle application using the Eclipse Kuksa software platform?* is the following. From the analysis of the literature and the author's experience in developing a vehicle navigation application using the Eclipse Kuksa software platform, the key advantages are as follows.

Eclipse Kuksa provides open source solutions that break the automotive industry's long-term closed island development model. The introduction of electronic systems 60 years ago has brought tremendous changes in the automotive industry. Software has become the primary source of automotive innovation (Mössinger, 2010). To improve their innovation capabilities, automotive OEMs began to invest heavily in their R&D. Increasing innovation, and cost pressures make them look for external resources to enhance their innovation capabilities. Ili, Albers, and Miller (2010) show that open innovation can provide better R&D productivity to car companies and be more suitable for the automotive industry than closed innovation. However, the status quo is still a closed innovation model. The Eclipse Kuksa in-vehicle and cloud platform takes open source as key elements. It is different from the development of proprietary solutions that are closed, i.e., with completely proprietary intellectual property codes, not available to the outside world, and can only run on certain types of cars. Eclipse Kuksa builds an open source software platform that can be added and reused, providing a universal application ecosystem for the developers. The author looked through the introduction, architecture, and source codes of the Eclipse Kuksa software platform; that opened information helped the author get to know the Eclipse Kuksa software development environment and then used it to do the implementation.

On the other hand, when faced with the IoT's challenge to the automotive industry, Eclipse Kuksa has established a vehicle-to-cloud solution standard and improved comprehensive development activities related to this field. The rapid rise of IoT and the rapid development of IoT technology have revolutionized the automotive market and industry. Buyers are now willing to wait for essential technical functions, connectivity,

and ease of use of cars. Consumers in connected vehicles push the automotive industry to transition from the product age to the service and experience era, from industry silos to complex interconnected ecosystems, and finally from connected cars to autonomous automotive industries (Krasniqi & Hajrizi, 2016). Automated driving cars use big data from a variety of built-in IoT devices. If people can't get a stable and reliable data stream, then autonomous vehicles will be useless on the road. The rise of autonomous vehicles is driving the need for connections to cloud platforms. OEMs face the challenge of requiring manufacturers to assume the role of a service provider. OEMs must deploy and manage these connections by creating IT professionals, software developers, and engineering teams to capture the market. Eclipse Kuksa adopts three-layer client–connection–cloud architecture for the internet of cars, connects vehicles to the cloud through in-vehicle and internet connections. It provides technologies that specifically solve the IoT, cloud, and digital vehicle system software's future design challenges. Eclipse Kuksa delivers an open and secure cloud platform supported by an integrated open source software environment. The Eclipse Kuksa cloud repository provides scripts and resources so that developers can set up their own Eclipse Kuksa cloud instances on the running Kubernetes instance. The challenges facing the internet of vehicles are highly concentrated in the areas of routing and communication protocols, security and privacy, data distribution, simulation, information management, and information fusion (Golestan et al., 2016). Eclipse Kuksa cloud utilizes Eclipse Hono, Eclipse hawkBit, Keycloak and many other IoT open source projects. It also uses InfluxDB, an open source database to collect, store and analyze vehicle data in the cloud. This thesis work used InfluxDB, and the vehicle location information was saved. Eclipse Kuksa cloud can also transmit information to solve the routing, communication protocol, security, data distribution, information management and integration challenges faced by the automotive industry. It provides manufacturers with an ecosystem foundation that can be used independently, as well as related protocols and communication technologies.

In addition, Eclipse Kuksa provides a collaborative development IDE: an efficient, scalable and business-friendly development environment for automotive applications and cloud development. It also provides automation for developing in-vehicle applications. It offers excellent convenience for developers. There are also applications and tools featured by Rover mobile robots to answer complex research questions. These applications and tools include cloud communication and open source tools. They are used in the Eclipse Kuksa cloud-based communication research to provide convenient services to analyze related technologies.

Eclipse Kuksa offers a complete and fully functional open source and security platform to tackle some of the long-term challenges of the exclusive software. It provides solutions to the automotive industry. However, use of the system still depends on the compatibility between the versions of software and hardware. Eclipse Kuksa integrates many IoT open source technologies. The current development of environment construction requires installing different software, which makes constructing the substrate environment complex. The incompatibility between the software and hardware versions causes a failure in the substate environment construction. In the beginning, the author tried to set up the Hono service and needed to use the old version of Hono to build the system, but failed to create the Hono service due to incompatible versions of the software. Furthermore, the system lacks a complete and fully functional technical introduction document. Users need to go to the Eclipse Kuksa website or GitHub to find the relevant parts for in-depth understanding. It requires lots of clicking on different links to search, read, and enter. In particular, the lack of compatibility between software and hardware versions has caused significant obstacles to users who are not familiar with related

technologies and environments. It would be a significant improvement if an installation package could automatically install the required software and hardware support.

Also, there is a lack of technical communication forums. Eclipse Kuksa brings together many open source IoT technologies. Will encounter various problems during development. Where can users ask and look for answers when there are questions? It is also related to the promotion of Eclipse Kuksa and continuous technical support. The forums required technical supporters familiar with Eclipse Kuksa to provide online technical support.

Finally, it needs to explain that Eclipse Kuksa is a solution developed for the automotive industry. Users are required to have software development experience and related background knowledge to use the system for design and development. It is a tremendous challenge for non-professional enthusiasts who use open-source software. Recommend reading the literature and related materials to obtain relevant background knowledge of related technical research.

6. Conclusion

This paper aims to study the development of vehicle navigation applications in Eclipse Kuksa and gain an insight into the benefits and challenges of using it for developing vehicle applications. The analysis used the DSR method to implement the vehicle navigation application through the DSRM Process to evaluate the development in Eclipse Kuksa.

Through the literature, the development of navigation applications and of the automotive industry and related challenges were studied. The background of Eclipse Kuksa and Roverapp for research implementation was also investigated. The designed system is a GPS-based vehicle navigation application based on Eclipse Kuksa. It realizes the requirements derived from section 4.1. These requirements are formulated based on the literature and the Eclipse Kuksa environment. The design phase of DSR is carried out following the DSRM Process to implement, including the planning of the GPS-based vehicle navigation application system based on Eclipse Kuksa. The required architecture, as well as the architecture design and workflow of the MQTT protocol were implemented by the network. The author also carried out development of applications on the vehicle and cloud platform and evaluate the achieved status.

The results show that the goal of the research has been achieved. By following the steps in the DSR, the DSRM Process implementation demonstrated how to develop a GPS-based vehicle navigation applications in Eclipse Kuksa, thus answering the first research question and verifying the feasibility of Eclipse Kuksa to develop vehicle applications. Chapter 5 discussed the benefits and challenges of developing vehicle applications in Eclipse Kuksa, thereby answering the second research question, contributing research to the knowledge base. Eclipse Kuksa offers a complete architecture of the automotive-centric IoT ecosystem. An open source automotive IoT-based solution to deal with the current IoT technology gap has not fully met the needs of automobiles. It provides a window for the automotive software ecosystem to adapt, combine and expand its project development to meet the future needs. This research is also a supplement to the use-cases of Eclipse Kuksa to develop vehicle applications. The connection of a vehicle to the cloud-based on the MQTT protocol supplements the source code implemented by Eclipse Kuksa based on the Hypertext Transfer Protocol (HTTP) protocol. The source code is for future developers in this field. It is another contribution research to the knowledge base.

Limitations and Future Research

Eclipse Kuksa provides a novel and comprehensive development platform for facing future challenges. On the other hand, it is a complex development platform covering many aspects. This paper is limited to the research between the in-vehicle and cloud platforms of Eclipse Kuksa. The app IDE is not involved. Choose the vehicle positioning, the common use of a vehicle, to do this research. Other parts of the in-vehicle platform have not been studied. The connection between the client and the cloud is studied through the MQTT protocol. There are other different protocols not reviewed. Reviewing this research, it was missing the Hono gateway implementation feature, one of Eclipse Kuksa's advantageous features. It could be the first future enhancement work to be done when the cloud server is available. Furthermore, this study selected the location of GPS. However, this was only due to time constraints. In the future, applications can add speed parameters and transmit them to the cloud platform. That is another exciting future research topic - to monitor whether the driver's driving behavior is correct and send a speeding message to remind the driver via SMS. Interested parties could also explore

other sensors on Rover, such as a temporary sensor that can provide weather reports to the driver. All these features could be developed based on the procedure of this paper. In addition, as mentioned earlier in this paragraph, the parts that are not involved in this research can be used as the subject of future research.

This study demonstrates that the advantages of using Eclipse Kuksa for the development of the automotive industry greatly outweigh the challenges. Research on this topic should continue, because it is a feasible solution for the automotive industry for facing future IoT challenges, filling the current gap in the field. Continued research and development in the academic and commercial fields will thus benefit many users in the automotive field.

References

About Kuksa. (2018). Retrieved Sep 4, 2019, from Kuksa: https://www.eclipse.org/kuksa/about/

About Eclipse-Kuksa. (2018). Retrieved Sep 4, 2019, from Eclipse-Kuksa: https://projects.eclipse.org/proposals/eclipse-kuksa

Appstacle. (2016). Retrieved Sep 2, 2020, from Appstacle: https://itea3.org/project/appstacle.html

Auto Connected Car. (2014). Retrieved Jan 21, 2020, from https://www.autoconnectedcar.com/definition-of-connected-car-what-is-the-connected-car-defined/

Automotive Grade Linux. (2016). Retrieved Sep 2, 2020, from https://www.automotivelinux.org/

Ayres, G. (2018, September 17). Driving Transformation: How Connected Vehicles Challenge OEMs. Retrieved November 30, 2020 from https://www.ibm.com/blogs/think/2018/09/driving-transformation-how-connected-vehicles-challenge-oems/

Bassil, Y. (2012). A simulation model for the waterfall software development life cycle. *arXiv preprint arXiv:1205.6904*.

Botta, A., De Donato, W., Persico, V., & Pescapé, A. (2016). Integration of cloud computing and internet of things: a survey. *Future generation computer systems*, *56*, 684-700.

Buehler, R. (2018). Can public transportation compete with automated and connected cars?. *Journal of Public Transportation*, *21*(1), 2.

Chadil, N., Russameesawang, A., & Keeratiwintakorn, P. (2008, May). Real-time tracking management system using GPS, GPRS and Google earth. In *2008 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology* (Vol. 1, pp. 393-396). IEEE.

Coppola, R., & Morisio, M. (2016). Connected car: technologies, issues, future trends. *ACM Computing Surveys (CSUR)*, *49*(3), 1-36.

Dhall, R., & Solanki, V. (2017). An IoT Based Predictive Connected Car Maintenance. *International Journal of Interactive Multimedia & Artificial Intelligence*, *4*(3).

Eclipse App4mc. (2020). Retrieved Sep 20, 2020, from https://www.eclipse.org/app4mc/

Eclipse Hono. (2019). Retrieved Sep 10, 2020, from https://www.eclipse.org/hono/

Eclipse Kuksa. (2020). Retrieved Sep 8, 2020, from https://www.eclipse.org/community/eclipse_newsletter/2020/january/1.php

Eclipse Paho. (2020). Retrieved Sep 18, 2020, from https://www.eclipse.org/paho/

Elliott, D., Keen, W., & Miao, L. (2019). Recent advances in connected and automated vehicles. *Journal of Traffic and Transportation Engineering (English Edition)*.

Golestan, K., Soua, R., Karray, F., & Kamel, M. S. (2016). Situation awareness within the context of connected cars: A comprehensive review and recent trends. *Information Fusion*, *29*, 68-83.

Harrold, M. J. (2000, May). Testing: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 61-72).

Hevner, A. R. (2007). A three cycle view of design science research. *Scandinavian Journal of Information Systems,* 19(2), 4.

Hevner, A., & Chatterjee, S. (2010). *Design Research in Information Systems, Theory and Practice*. Springer.

Ibrahim, M., Elgamri, A., Babiker, S., & Mohamed, A. (2015, October). Internet of things based smart environmental monitoring using the Raspberry-Pi computer. In 2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC) (pp. 159-164). IEEE.

Ili, S., Albers, A., & Miller, S. (2010). Open innovation in the automotive industry. R&d Management, 40(3), 246-255.

IoT Kuksa. (2020). Retrieved Sep 8, 2020, from https://projects.eclipse.org/projects/iot.kuksa

Jackson, B. M., Polglaze, T., Dawson, B., King, T., & Peeling, P. (2018). Comparing Global Positioning System and Global Navigation Satellite System Measures of Team-Sport Movements. *International journal of sports physiology and performance*, *13*(8), 1005-1010.

Järvinen, P. (2004). *On research methods*. Tampere: Opinpajan kirja.

Kamel, M. B. M. (2015). Real-time GPS/GPRS based vehicle tracking system. *International Journal Of Engineering And Computer Science*, *4*(08).

Krasniqi, X., & Hajrizi, E. (2016). Use of IoT technology to drive the automotive industry from connected to full autonomous vehicles. *IFAC-PapersOnLine*, *49*(29), 269-274.

Kuksa documentation. (2019). Retrieved Sep 10, 2020, from https://www.eclipse.org/kuksa/documentation/

Lee, S., Tewolde, G., & Kwon, J. (2014, March). Design and implementation of vehicle tracking system using GPS/GSM/GPRS technology and smartphone application. In *2014 IEEE world forum on internet of things (WF-IoT)* (pp. 353-358). IEEE.

Li, X., Zhang, X., Ren, X., Fritsche, M., Wickert, J., & Schuh, H. (2015). Precise positioning with current multi-constellation global navigation satellite systems: GPS, GLONASS, Galileo and BeiDou. *Scientific reports*, *5*, 8328.

Madakam, S., Ramaswamy, R., & Tripathi, S. (2015). Internet of Things (IoT): A literature review. *Journal of Computer and Communications*, *3*(05), 164.

Maksimović, M., Vujović, V., Davidović, N., Milošević, V., & Perišić, B. (2014). Raspberry Pi as Internet of things hardware: performances and constraints. *design issues*, *3*(8).

Maurya, K., Singh, M., & Jain, N. (2012). Real time vehicle tracking system using GSM and GPS technology-an anti-theft tracking system. *International Journal of Electronics and Computer Science Engineering. ISSN*, *22771956*, V1N3-1103.

Muruganandham, P. M., & Mukesh, R. (2010). Real time web based vehicle tracking using GPS. *World Academy of Science, Engineering and Technology*, *61*(1), 91-9.

Mössinger, J. (2010). Software in automotive systems. *IEEE software*, *27*(2), 92-94.

Obradovic, D., Lenz, H., & Schupfner, M. (2007). Fusion of sensor data in Siemens car navigation system. *IEEE Transactions on Vehicular Technology*, *56*(1), 43-50.

Pakanen, O. P., Ahmad Banijamali, A., Haghighatkhah, O. L., Destino, G., Kuvaja, P., Latva-aho, M., ... & Laaroussi, Z. Breaking the Silos in Automotive Software and Systems Development.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, *24*(3), 45-77.

Petrov, C. (2020, October 13). 47 Stunning Internet of Things Statistics 2020 [The Rise Of IoT]. Retrieved November 30, 2020 from https://techjury.net/blog/internet-of-things-statistics/#gref

Piao, J., Beecroft, M., & McDonald, M. (2010). Vehicle positioning for improving road safety. *Transport reviews*, *30*(6), 701-715.

Priyadharshini, M. D., Ponmurugan, P., Nishanthi, M., & Elzalet, J. (2019). EFFICIENT DEVICE FOR MEASURING AND CONTROLLING VARIOUS PARAMETERS IN STANDALONE SOLAR/WIND SYSTEM INCORPORATING INTERNET OF THINGS.

Rahiman, W., & Zainal, Z. (2013, June). An overview of development GPS navigation for autonomous car. In *2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)* (pp. 1112-1118). IEEE.

Ramani, R., Valarmathy, S., SuthanthiraVanitha, N., Selvaraju, S., Thiruppathi, M., & Thangam, R. (2013). Vehicle tracking and locking system based on GSM and GPS. *IJ Intelligent Systems and Applications*, *9*, 86-93.

Rover intro. (2017). Retrieved Sep 17, 2020, from https://app4mc-rover.github.io/rover-docs/content/intro.html

Shuttleworth, J. (2019, January 7). SAE Standards News: J3016 automated-driving graphic update. Retrieved December 9, 2020, from https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic

Structure of Kuksa. (2018). Retrieved Sep 4, 2019, from https://www.eclipse.org/community/eclipse_newsletter/2018/july/kuksa.php

Tang, H., Shi, J., & Lei, K. (2016, June). A smart low-consumption IoT framework for location tracking and its real application. In *2016 6th international conference on electronics information and emergency communication (iceiec)* (pp. 306-309). IEEE.

Thomas, M. O., & Rad, B. B. (2017). Reliability evaluation metrics for internet of things, car tracking system: a review. *Int. J. Inf. Technol. Comput. Sci.(IJITCS)*, *9*(2), 1-10.

Vega, M. (2020, November 21). Internet of Things Statistics, Facts & Predictions [2020's Update]. Retrieved November 30, 2020 from https://review42.com/internet-of-things-stats/

Walker, J., & Awange, J. L. (2018). Global navigation satellite system. In *Surveying for Civil and Mine Engineers* (pp. 147-156). Springer, Cham.

Warrendale, PA. (2018, December 11). SAE International Releases Updated Visual Chart for Its "Levels of Driving Automation" Standard for Self-Driving Vehicles. Retrieved December 9, 2020, from https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles

Zhao, J., Liang, B., & Chen, Q. (2018). The key technology toward the self-driving car. *International Journal of Intelligent Unmanned Systems*.

Zito, R., D'este, G., & Taylor, M. A. P. (1995). Global positioning systems in the time domain: How useful a tool for intelligent vehicle-highway systems?. *Transportation Research Part C: Emerging Technologies*, *3*(4), 193-209.

Appendix A. Abbreviations

V2X	Vehicle-to-everything
IoT	Internet of Things
TCP/IP	Transmission Control Protocol/Internet Protocol
OEM	Original Equipment Manufacturer
IDE	Integrated Development Environment
APPSTACLE	open standard APplication Platform for carS and TrAnsportation vehiCLEs
GPS	Global Positioning System
DSR	Design Science Research
DSRM	Design Science Research Methodology
WLAN	Wireless Local Area Network
V2R	Vehicle-to-RSU (Road-side Units)
V2I	Vehicle-to-infrastructure
V2V	Vehicle-to-vehicle
V2H	Vehicle-to-human
V2S	Vehicle-to-sensor
VANET	Vehicular Ad-hoc Network
UMTS	Universal Mobile Telecommunications System
LTE	Long Term Evolution
CAV	Connected and automated vehicle
SAE	Society of Automotive Engineers
AV	Automated vehicle
HAV	Highly automated vehicle
GNSS	Global Navigation Satellite System
GLONASS	GLObal NAvigation Satellite System
GPRS	General Packet Radio Service
RFID	Radio Frequency Identification

WSN	Wireless Sensor Network
AGL	Automotive Grade Linux
ECU	Engine control unit
API	Application Programming Interface
MQTT	Message Queuing Telemetry Transport
SN	Sensor Network
HTTP	Hypertext Transfer Protocol