University of Gothenburg

DIT168

Project: Industrial IT and Embedded Systems

Software Architectural Document

Authors:
Erik Laurin
Isabelle Trnqvist
Joacim Eberlen
Justinas Stirbys

Group 01

March 30th, 2018

Contents

1 Revision History

The evolution of the Software Architectural Document for project dashFTABs is detailed under this section. Emphasis is put on changes incorporated, via Description column, the date and the author. In situation where all members have contributed to a change the author will be listed as Group 01.

Date	Version	Description	Author
27th March, 2018	0.1	Added Functional Requirements	Group 01
4th April, 2018	0.2	Added Introduction	Justinas
5th April, 2018	0.3	Added Use Case View	Justinas
12th April, 2018	0.4	Added Sequence Diagrams for UC1-UC3	Justinas
18th April, 2018	0.4.3	Added 4+1 Vew Model to Introduction, UC1-UC3 sequence diagram descriptions to Process View	Justinas
21st April, 2018	0.5	Added System Context	Erik
17th May, 2018	0.6	Added Logical View (from "Conceptual Ideas")	Isabelle
20th May, 2018	0.7	Added Final Report subsection in Introduction, Drive Car Use Cases (UC), UC Significance and Behaviour subsections for Connect as Leader and Stop Following, and references for external libs	Justinas
20th May, 2018	0.8	Added Physical View (from "Hardware & Software Integration)	Justinas
20th May, 2018	0.9	Added Maneuvering subsection to Process View (from "Algorithmic Fundamentals")	Erik
20th May, 2018	1.0	Added Development View (originally "Software Architecture" section)	Joacim

The rows coloured purple indicate parts of the document that were taken from the Final Report[1]. Specific sections of the Report are written in parenthesis. The only section to be taken fully from the Final Report is the Development View.

2 Introduction

2.1 Purpose

The purpose of this document is to familiarize the reader with the architectural overview of the software, developed for the project dashFTABs, by examining different architectural viewpoints. In continuation, the document includes architectural drivers, such as system requirements, and attempts to identify significant dashFTABs systems behaviour.

2.2 Scope

The software was developed as part of the DIT168 Project: Industrial IT and Embedded Systems course taught at University of Gothenburg in Gothenburg, Sweden. The project course provides a 3D printed miniature car, dubbed Dash by the project group. The project groups communicate amongst each other to implement a Vehicle-to-Vehicle (V2V) protocol. Moreover, via incorporation of the protocol the project groups must expand Dashs functional capabilities by implement platooning.

2.3 Audience

The document contains technical details of the project and utilizes Unified Modeling Language (UML for short) to express architecture. Therefore, the reader must possess some minimal or introductory knowledge of UML. As such the intended readers are members or somewhat affiliated with software engineering.

2.4 4+1 View Model

To further narrow down the intended audience the document incorporates 4+1 View Model. The document will focus on architectural logical, process, development and physical views. With each view providing value to specific set of stakeholders. Logical View provides most benefit to the system's end-users as it addresses whether all desired functionality has been implemented; the Process View focuses on the overall system by depicting the subsystems, meaning microservices, interactions; Development View is aimed at developers as it focuses more on individual module contents; lastly,

Physical View shows the physical distribution of software, thus it provides most value to system engineers.

The "+1" of the the 4+1 View Model refers to Scenarios, which are used to describe the system behaviour. The Scenarios provide value to all stakeholders as it attempts to encapsulate the system requirements. The Scenarios have been expressed as a Use Case View.

2.5 Final Report Overlap

The document has some overlap with the Final Report[1], as shown by the Revision History, Section 1. The specified overlaps were initially written for the Final Report. However, they were of interest to the Software Architectural Document. Moreover, the overlapping sections were planned to be included in the SAD before the Final Report had been written. This can be seen by the previous "4+1 View Model" subsection.

3 System Context

3.1 Project Context

Putting our software on the miniature car provides the user of the car with two main features; the ability to maneuver the car and the ability to make the car follow another car of the same type. The user controls, and maneuvers, the car through a web-based interface, the web remote controller, where one also can engage the car's following mode which shall result in the car starting to follow another.

A simple domain model was created further improve the understanding of data flow and the interactions between different system parts.

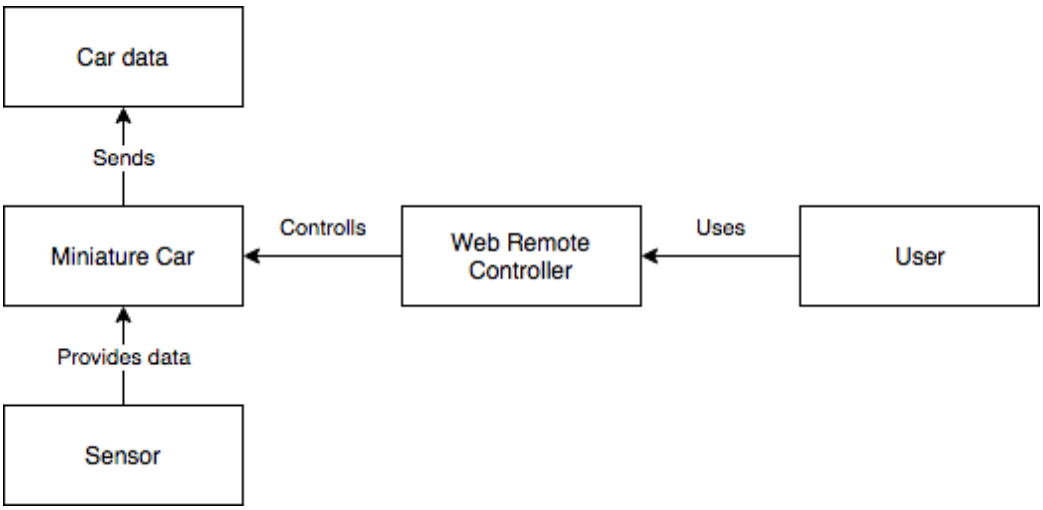

Figure 1: Domain Model

3.2 System Interface

The software provides the user two way of interfacing with the car; the so called terminal remote controller and the web remote controller.

The terminal remote controller provides the user with the ability to control the car through a simple computer terminal. Here, the user controls the car through the use of the keys. Furthermore, the user may also set the speed and the steering angle car should have when driving. Lastly, in case an obstacle is detected within a set distance, the car will break.

The more extensive web remote controller provides aforementioned features, (apart from being able to set the steering angle and distance before breaking for obstacles) together with another set of functionalities. Through the web remote controller, the car's speed, heading, distance traveled and the distance to obstacles are presented. In addition, most data that the car sends, receives and processes are displayed in the interface. Lastly, the interface provides the user with a switch that chooses which mode the car shall be in; leader (the user controls the car) or follower (the car follows another car autonomously).

3.3 External Libraries/Images

To realize our software, various libraries and Docker images were used.

Name	**Description**
libcluon[2]	Christian Berger's C++ library libcluon is the hub of our software. It is used for all communication, both internally in the car and externally to other car's.
seresearch/ 2018-dit-168	From this image we run two microservices, odsupercomponent and the proxy-miniature-pwm-motor which allow our software to control the miniature car through sending messages with the libcluon library.
opendlv-device-ultrasonic-srf08-armhf	This image gets the front ultrasonic sensor's reading and make it available for our code to use.
Robotics Cape[4]	This library allows our software to access data readings from the sensors that are placed on the BeagleBone.
opendlv-signal-viewer[5]	This library was used as the base of the built web interface

4 Architectural Drivers

4.1 Functional Requirements

Functional requirements were used to identify and narrow down the project scope. The requirements are prioritized using MoSCoW notation i.e. requirements are divided into Must, Shoulds, Coulds, and Wonts. Must dictates requirements that are mandatory for the final demonstration. Should expresses requirements that are significant, but do not have a defined deadline. Could expresses requirements/features that would improve the project quality, but are not necessarily implemented. Lastly, Wont is used to track identified requirements that will not be implemented, due to product owner dislike or disapproval, time and budgetary constraints.

ID	Requirement	Description	Status	Priority
F1	Message Log	A web page must contain a message log of everything that has been sent internally and externally within the car	Implemented	Must
F2	Remote Controller	A web page must contain a graphical remote controller that communicates and controls Dash, when the car is the leader of the platoon	Implemented	Must
F4	Leader Connection	The car, Dash, must be able to support Leader functionality (i.e. send LeaderStatus requests) while platooning	Implemented	Must
F5	Follower Connection	The car, Dash, must be able to participate in platooning as a follower	Implemented	Must
F6	Maneuvering	The car will drive forward, turn left or right on commands received over the OD4 session	Implemented	Must
F7	IMU	Dash must be able to use the IMU on its BeagleBoard Blue to calculate the distance moved	Implemented	Must

F8	V2V Protocol	The car must be able to support the V2V Protocol. It is required for it to communicate with other cars and send sensors data	Implemented	Must
F9	Collision Prevention	Dash will stop/brake when ultrasonic readings return an object that is less or equals to 10 cm ahead	Implemented	Should
F10	Emergency Brake	The car will stop if it fails to receive 3 update requests (i.e. hasnt received anything in 300ms) and/or the connection to other cars has been lost	Implemented	Must
F11	Video Streaming	The car could incorporate the RPi and camera to live stream its video	Not Implemented	Could
F12	Lane Following	Via incorporation of the RPi camera, Dash could implement identification and following of straight lines	Not Implemented	Could

5 Use Case View

This document section attempts to identify the main behaviour exhibited by Dash. This is done by expressing the behaviour as use case scenarios. The chosen scenarios depicts some of the significant features of developed project. Moreover, the identified scenarios all utilize the V2V Protocol in different aspects, meaning the use case scenarios use different messages from the protocol. Therefore, the identified scenarios fulfil different aspects of the functional requirements of F8 (V2V Protocol).

5.1 Use Case Scenarios

5.1.1 Connect As Follower

ID	UC1
Use Case	Connect as Follower
Description	Dash connects to another miniature car as a Follower and begins to receiving Leader Status update messages
Actors	Dash, Another Car
Preconditions	Excluding Dash there is another miniature car with the V2V protocol that are not already connected to any cars

Steps	Basic Flow: 1. Dash uses Announce Presence (its not very effective). Another car receives Dashs IP and group number 2. Another Car announces presence. Dash receives its IP and group number 3. Dash uses Follow Requests. Selects which group car to follow 4. Another Car receives follow request and does not have a follower yet. Another Car sends Follow Response 5. A connection using UDP Sender and Receiver is established between Dash and Another Car Alternative Flow: 4. Another Car already has a Follower. (a) No Follower Request is sent (b) Display a message informing the users (c) Continues at Basic Flow 2

5.1.2 Connect As Leader

ID	UC2
Use Case	Connect as Leader
Description	Dash connects to another miniature car as a Leader and begins to sending Leader Status update messages
Actors	Dash, Another Car
Preconditions	Excluding Dash there is another miniature car with the V2V protocol that are not already connected to any cars

Steps	Basic Flow: 1. Dash uses Announce Presence (its not very effective). Another car receives Dashs IP and group number 2. Another Car announces presence. Dash receives its IP and group number 3. Dash receives follow request and does not have a follower yet. Dash sends Follow Response 4. Another Car receives follow request and does not have a follower yet. Another Car sends Follow Response 5. A connection using UDP Sender and Receiver is established between Dash and Another Car Alternative Flow: 4. Dash already has a Follower. (a) No Follower Request is sent (b) Display a message informing the users (c) Continues at Basic Flow 2

5.1.3 Stop Following Leader

ID	UC3
Use Case	Connect as Leader
Description	Dash disconnect from Another Car, which acted as a Leader
Actors	Dash, Another Car
Preconditions	Dash is already connected as a Follower to Another Car

Steps	Basic Flow: 1. Dash sends Follower Status and receives Leader Status, indicating its following Another Car 2. Dash sends Stop Following message and stops moving 3. Dash removes Another Car as its leader 4. Another Car receives stop following message. Another Car removes Dash as its follower

5.1.4 Drive Dash

The Drive Dash is comprised of 2 Use Cases. The first one depicts the scenario when Dash is leading another vehicle, the latter when Dash is being lead by another miniature vehicle.

ID	UC4
Use Case	Drive as Leader
Description	Another Car connects to Dash as a Follower, at which point a Driver uses the remote controller to maneuver Dash
Actors	Dash, Driver, Another Car
Preconditions	Another Car is already connected to Dash as a Follower
Steps	Basic Flow: 1. The Driver enables "Leader Role" through web interface. 2. Dash swaps to leader mode. 3. The Driver uses "WASD" keys to control Dash. Dash moves. 4. Dash sends LeaderStatus to Another Car Alternative Flow: 3. Dash detects an object ahead (a) Dash stops

ID	UC5
Use Case	Drive as Follower
Description	Dash connects as a Follower to Another Car and begins autonomous moving
Actors	Dash, Driver, Another Car
Preconditions	Dash is connected as Follower to Another Car
Steps	Basic Flow: 1. The Driver enables "Follower Role" through web interface. 2. The "WASD" keys used for moving are disabled. 3. Dash swaps to follower mode. 4. Dash receives LeaderStatus from Another Car. 5. Dash queues LeaderStatus and begins driving using "speed" from LeaderStatus. 6. Maximum queue size is met. Dash begins turning with "steeringAngle" from LeaderStatus that were queue. 7. Dash removes top item from queue. Alternative Flow: 5. Dash is not in follower mode (a) Dash ignores messages (b) Retry Basic Flow 2

5.2 Use Case Diagram

To depict use case interactions amongst each other, a use case diagram was created.

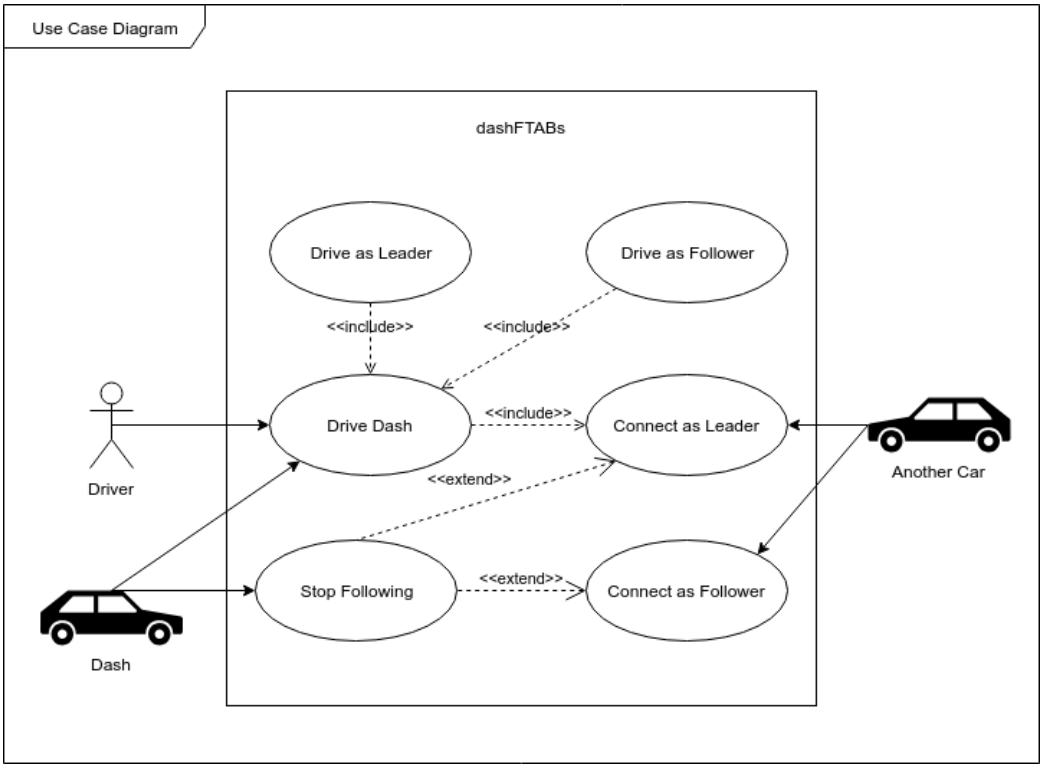

Figure 2: Use Case Diagram

The diagram also servers as a visual aid, aimed to improve understanding of actor interactions with project functionality, as such it could also be included in the Logical View. However, it was believed its best served to tie in together all established Use Cases, as the diagram depicts the interactions between the most significant use case scenarios, mentioned in previous sections.

6 Process View

The Process View section aims to inform the reader on subjects such as communication between the designed software components and communication between the project's subsystem i.e. the developed microservices. The section graphically express the architecturally significant use cases (See section Use Case View). Sequence Diagrams (SD for short) were used for use case realization, while natural language was used to improve the readers understanding of the architectural significance the use cases provide.

6.1 V2V Protocol

As the focus for the project is vehicle-to-vehicle (V2V) communication, a shared V2V protocol was developed by a representative from each team for the purpose of being utilized in the project. The project group dashFTABs has altered the provided protocol to accommodate their team's needs. The changes introduced do not go against any established agreements as the protocol is licensed under GNU General Public License v3.0, a license allowing modification. Moreover, the protocol was design in a way that would require small alterations to accommodative individual teams architecture. Due to the protocol being used by all project groups the behaviour exhibited by Connect as Leader, Connect as Follower will be similar across all teams.

The specific changes to the protocol include; creating an additional OD4 session for internal car communication, broadcasting the messages received from other cars to the internal channel, as well as an additional function responsible for monitoring time Leader/Follower Status request have been received.

6.2 Connect As Follower

6.2.1 Significance

The use case was chosen as architecturally significant, as it implements functional requirement F5, Follower Connection. The requirement itself was provided by the DIT168 course representatives, as it is mandatory for every miniature vehicle to posses follower capabilities. Without a follower connection, Dash would not be able to have automated moving or platooning, as it would not receive status updates from the leader. Therefore, F5 requirement

acts as a prerequisite to more advanced functionality. Due to these reasons the requirement was deemed as significant and expressed as a sequence diagram. The sequence diagram depicts the behavior of UC1.

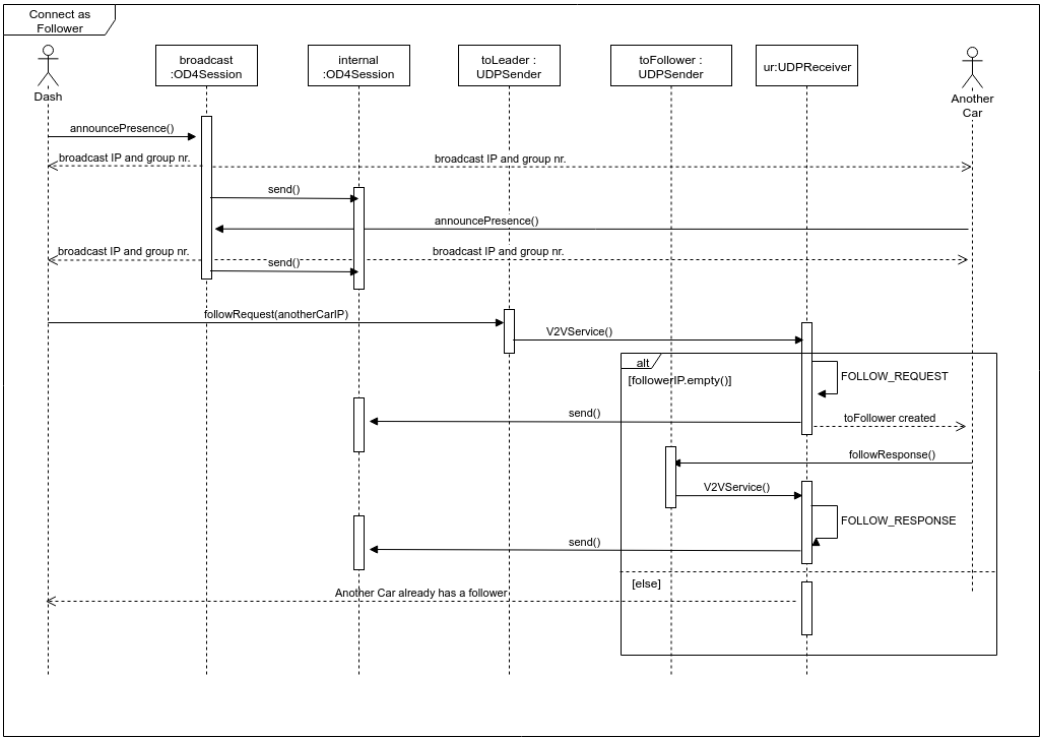

Figure 3: Connect As Follower SD

6.2.2 Behaviour

The interactions depicted in Figure 3 focus specifically on behaviour exhibited by the V2V microservice. The diagram consists of 5 main components; OD4 Sessions broadcast and internal, UPD Senders toLeader, toFollower, and receiver a UDP Receiver. The broadcast OD4 session listens to CID 250, a channel used for cars that wish to engage in platooning. Once a car has used announcePresence() their car IP and group number are broadcasted to everyone on the network. Dash then send a FollowRequest, that is evaluated on the UDP Receiver. If Another Car does not have any followers it send a FollowResponce and uses Dasd's IP to establish a toFollower channel.

Once Dash receives the FollowResponce message it creates a toLeader UDP Sender and the message exchange can begin. It is worth stating taht after every message is sent or received it is forwarder to "internal" OD4 session for data visualization.

6.3 Connect As Leader

6.3.1 Significance

Similarly to UC1, this use case was chosen for being a precursor to more advance behavior. To fulfil the project requirement the car must be able to create a connection as Leader and send LeaderStatus to realize V2V following.

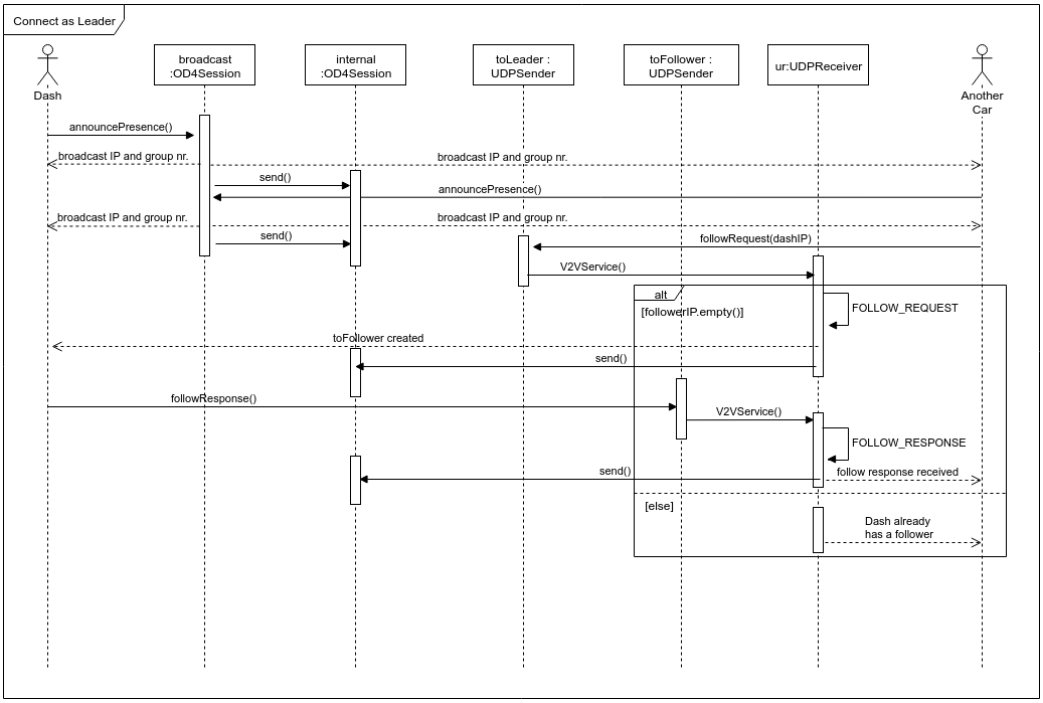

Figure 4: Connect As Leader SD

6.3.2 Behaviour

The interactions begins with both car announcing their presence, to obtain the cars' IPs and IDs. Once this is done Another Car sends a FollowRequest and Dash responds with FollowResponce or ignoners the message in case Dash already has a follower. All messages are forwarded to the internal communication channel as well. The behaviour is identical to the one seen in section 6.2.2.

6.4 Stop Following

6.4.1 Significance

This use case is the least significant out of the established use cases. It was chosen as it realizes the logic for closing established connection between the cars, a mandatory behaviour from the V2V-Protocol. This is the only use case depicting or incorporating this type of behaviour.

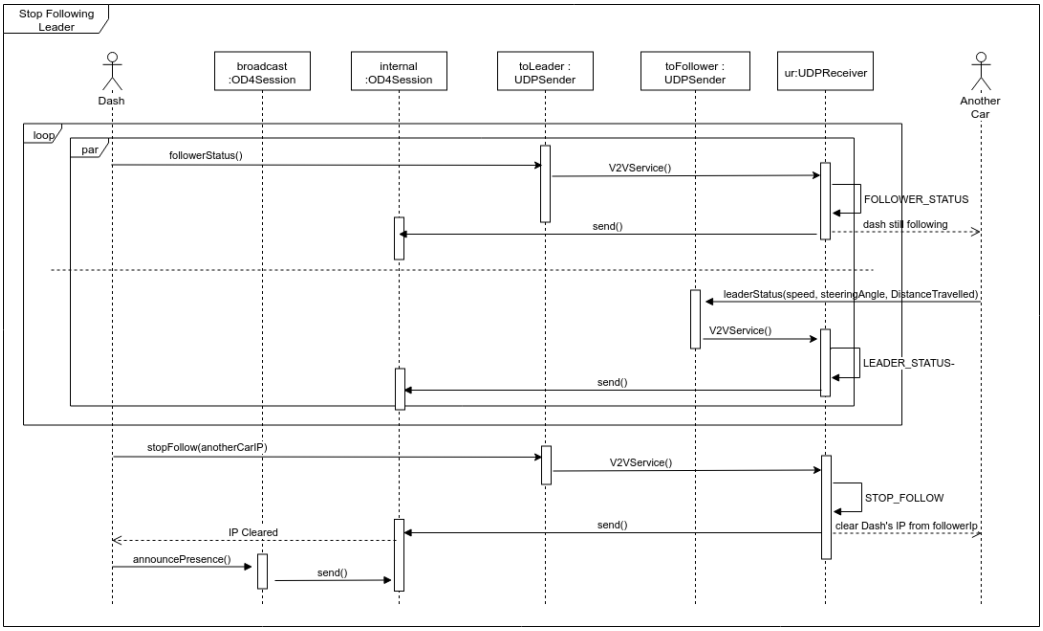

Figure 5: Stop Following SD

6.4.2 Behaviour

This use case scenario requires the connections to be established and messages sent. This can be seen via the loop in the sequence diagram. The "par" box is used to depict the messages being sent in parallel. Dash send the StopFollow message to close the UDP connections and wipe the IP. It is important to mention that the exact same behaviour applies regardless of Leader/Follower role.

6.5 Maneuvering

6.5.1 Importance

The maneuvering behavior in divided into two parts. The first is driving as a Leader, the second driving as a Follower. The combined action of these two behaviours realize the "Drive Dash" user case, see section 5.1.4.

6.5.2 Behaviour

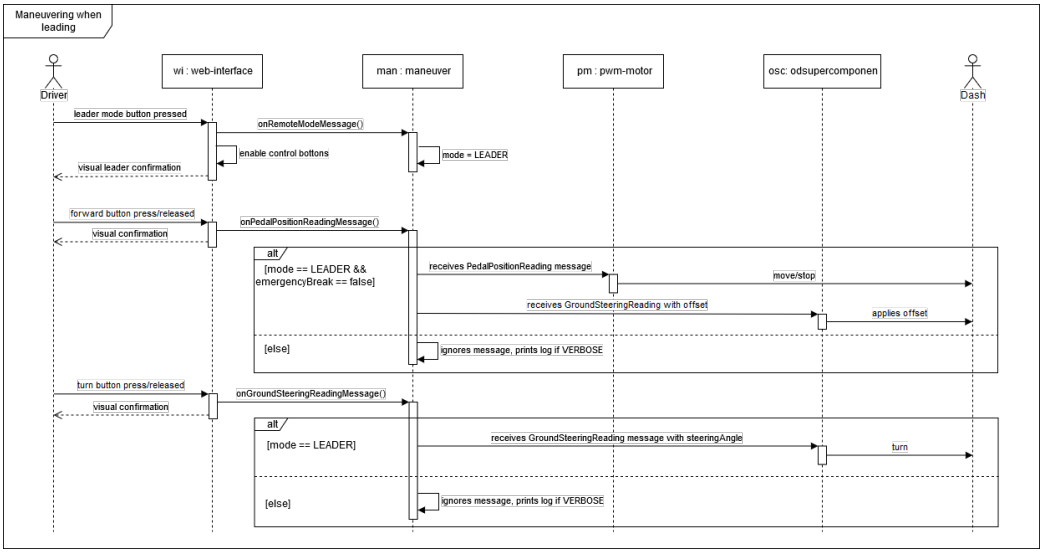

Figure 6: Sequence Diagram - Leader

The diagram above depicts the chain of events when controlling the car through the web-interface. Note that turn and forward buttons can be

pressed without any particular order. However, leading mode must be enabled for the microservice to evaluate the commands. In case leading mode is enabled and emergencyBreak is false (true in case an obstacle is detected by ultrasonic sensor within a specified distance), commands to move or stop will be forwarded to the pwm-motor microservice. Lastly, commands to turn the servo and thus the car is directly forwarded to the odsupercompontent as long as leading mode is activated.

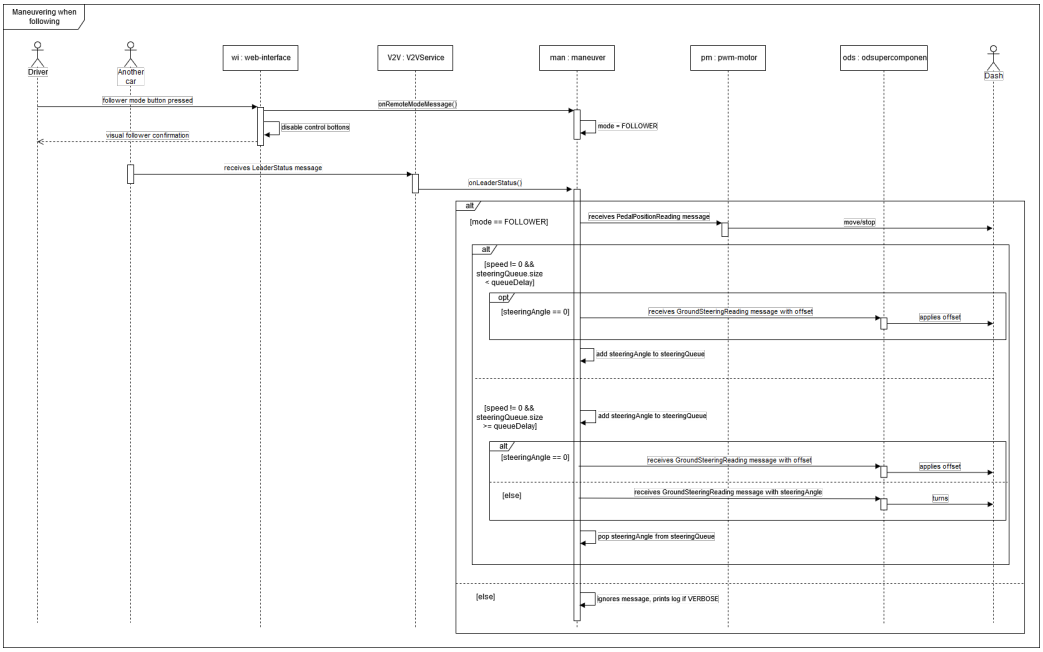

Figure 7: Sequence Diagram - Follower

Similarly to the first sequence diagram, the diagram above illustrates the chain of events when the car is executing commands. However, instead of having a driver controlling the miniature vehicle directly through the web-interface, the car is following another miniature car autonomously. This is made possible through the retrieval of so called 'LeaderStatus' messages from the leading vehicle. This message is evaluated and processed and eventually a command is sent to the microservices odsupercomponent and pwm-motor for the car to move.

7 Logical View

This section aims to provide a logical overview of the overall structure of the product. The intended audience for this section is someone seeking fundamental understanding of the developed system, and will therefore describe necessary architectural and design decisions on a high level.

Visualized in Figure 8 are the relevant components and their connections in the dashFTABs system.

'odsupercomponent' and 'pwm-motor' both are components that were provided to the development team by the product owner. The use of microservices are consistent throughout, and was a considered decision made in the inception of the project. For the rationale of this decision, please refer to 8.7.

The communication line between all components shown here makes use of the Libcluon library, also provided by the project's product owner. The message exchange is done through 'OD4 sessions': utilizing channels and assigning ID's to messages, in this context referenced to as 'CID's. For further explanation of the channels, refer to 8.7.

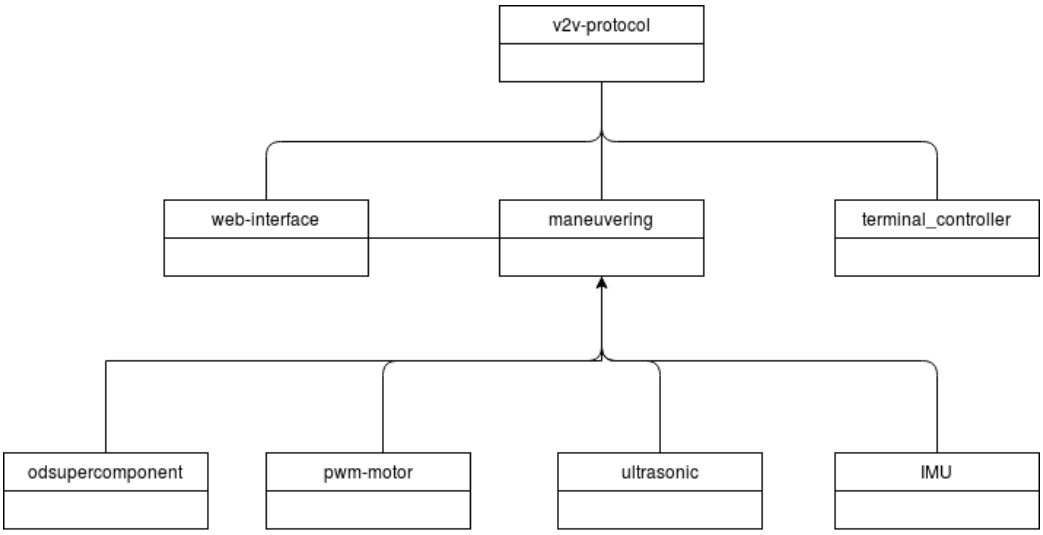

Figure 8: Simple Class Diagram

8 Development View

Deciding on architecture is a vital part of any project, thinking of what benefits a team can reap from the choice during the phases of development. During presentations by Dr. Rev. Christian Berger, it became quite obvious that their architecture of choice was microservices. The team had discussion about the pros and cons of this, see section 8.7.

One of the benefits of not choosing the microservice architecture was going for a more familiar architecture that the developers have experience working with. During early discussions, the development team did not find enough reason to not try the new way of designing the system.

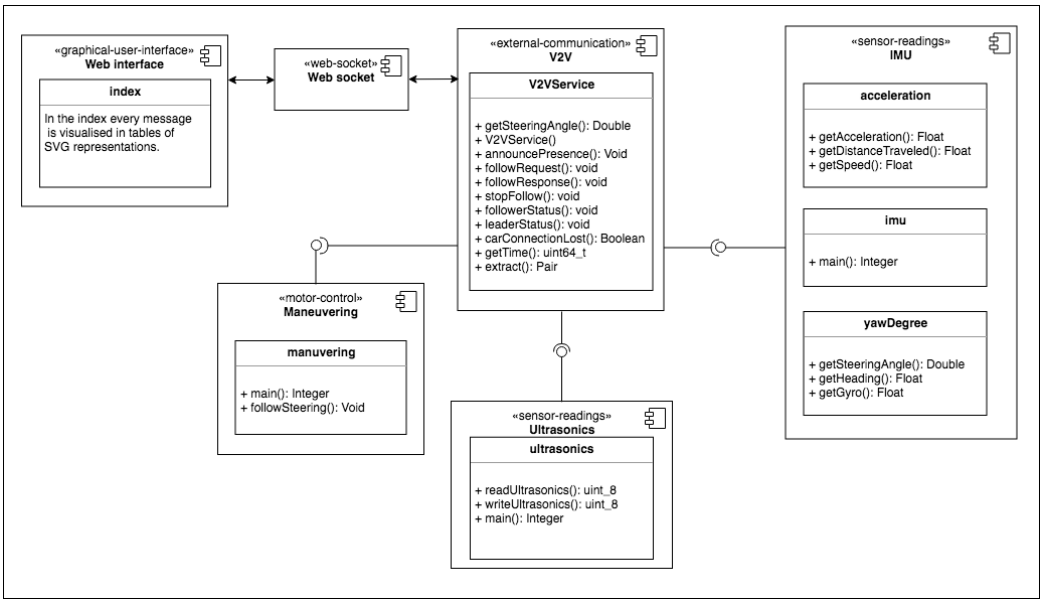

Figure 9: Structured class diagram

8.1 V2V Service

Designed with the V2V Service as a central node in our architecture we can easily route data throughout the system using different CID:s. The project uses multiple connection channels in the V2V Service:

1. Internal [Channel 122] This channel is used by the system for internal communication. The internal channel was a design decision based on

scalability of our system. Utilizing this channel every part of our system can easily access the data needed to visualize or execute actions accordingly.

2. Broadcasting [Channel 250] This channel has some of the messages for the external systems in our environment. The external systems are composed out of other groups cars. The message, at the moment of writing this, using the channel is the announce presence. If there is a need to send more messages in this channel it can be developed further.

3. Incoming [UDP Connection] This UPD connection reroutes the incoming messages from other vehicles into the internal channel. When the messages have been distributed, the rest of the system can use the Internal channel OD4 session to access this data.

4. To Leader [UDP Connection] This channel is used to send messages to the leader car. This is based on the V2V protocol that all groups use, the protocol gives us a straightforward implementation to exchange data between the cars.

5. To Follower [UDP Connection] The follower channel sends status updates to the follower car, the messages sent through this channel gives a uniform understanding of the current position of the cars of interest.

8.2 IMU

To use the IMU on the BeagleBone board some calculations were needed to be conducted. This microservice has the calculations of the returned data from the IMU hardware component.

The IMU is used to get the change in heading, distance traveled and speed of the car. It broadcasts the data through an enveloped message to the internal channel.

8.3 Maneuvering

The maneuvering microservice is the connection to the actual hardware of the car, it sends the messages from the V2V microservice to control the vehicle. It is designed with two modes, one follower, and one leader. The mode is supposed to decide what messages are interesting to the miniature car.

8.4 Ultrasonics

The ultrasonic microservice gives the distance in front of the car, it reads the values from the connected hardware component on the car. The rationale for this service was mainly the emergency brake event. The plan was to use a dynamic user-set distance, which should stop the car instantly. In the future, this could provide redundancy with other components to secure the execution of the emergency brake.

8.5 Web Interface

To control the car manually and visualize the internal working of the system, it was decided to use an extension of Chalmers Revere's signal viewer[5]. In this microservice, we integrated the manual controls as arrow keys and divided the messages into internal and external, to make the visualization a bit clearer. The external messages are everything sent by the V2V Protocol, whereas the internal messages are the data transfered with the project group's vehicle. The IMU and the ultrasonic sensor is represented in animated images, this gives the user a better experience when using the system.

8.6 Terminal Controller

As an interface for testing the car during the project, another microservice, the Terminal Controller, was used. It was developed for simplicity and stability. Initially, the web-interface was a bit lacking in performance and the terminal controller could be used as a backup. Having a backup microservice to use when needed increases the redundancy of the development process, but provides the system users with a backup remote controller that could be used for testing purposes and web-interface validation.

8.7 Rational for Microservices

Using the aforementioned microservices when working with an embedded system that contains not only our own services, but also microservices created by other developers, gave us a more modifiable system. When microservices are paired with Docker or other operating-system-level virtualization techniques, the whole system becomes fully portable. Moreover, as a whole, a shared vision and knowledge sharing between developers was easier to convey.

If the team were to continue developing the solution for the autonomous driving or graphical user interface this type of architecture has the power to scale seamlessly as an infinite number of additions can be developed with ease.

One of the bad things with the chosen architecture was, what should each of the services contain, how small should they be and how can we make sure there is no overlap in functionality in the system. In our case, the solution was partly to have a predefined process to ensure that aforementioned shortcomings were mitigated. Even if this process needed to be improved, the practice of dividing the work was carried over from earlier endeavours for the majority of our team.

When pressed for time, a microservice based architecture can be a large benefit due to the enabling of parallel development of the different services. Anything developed should be able to run independently from everything else in the system. This gave us the ability to test, format and refine each service without having to worry about any other developers work progress. Delivering for each deadline during the project was uncomplicated to monitor, everyone knew who needed to do what part and could help accordingly.

9 Physical View

A deployment diagram, as seen in Figure 10, was created to visualize the distribution of software amongst hardware. The software was deployed onto a BeagleBone. Most of the software developed was encapsulated within Docker images, with few exceptions.

The communication between the microservices is done using OD4 Sessions provided by libcluon[2], as such libcluon is either installed or included as a header only library in each microservice. Moreover, each microservice includes, at the minimum, a Messages.odvd file that defines message specification and an OD4 Session to transfer the readings to an internally defined OD4 channel, expressed with CID 122.

The "ultrasonic" and "imu" microservices were built as executables as the dependent libary [4] was not able to be encapsulated within docker images.

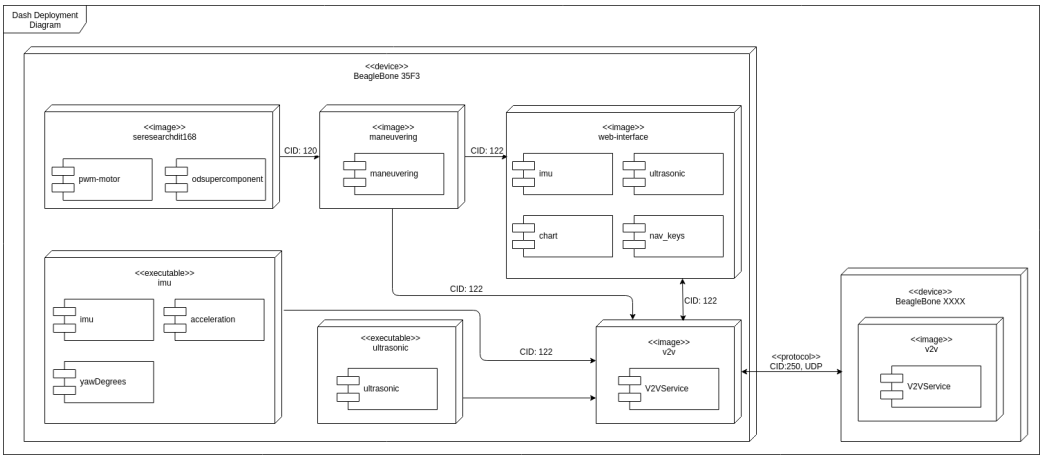

Figure 10: Deployment Diagram

10 References

[1] Laurin, E., Trnqvist, I., Eberlen, J., & Stirbys, J. Final Report, GitHub repository: https://github.com/justasAtGU/dit168/tree/master/doc/ProjectManagement/FinalReport

[2] Berger, C., libcluon, GitHub repository: https://github.com/chrberger/libcluon

[3] Chalmers Revere, GitHub repository; https://github.com/chalmers-revere/opendlv-device-ultrasonic-srf08

[4] Strawson Design, Robotics Cape, Website: http://www.strawsondesign.com/#!

[5] Chalmers Revere, GitHub repository: https://github.com/chalmers-revere/opendlv-signal-viewer