CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Software Architecture for Reducing Testing Effort while Ensuring Safety in Autonomous Road Vehicles

Master's thesis in Software Engineering and Technology

MARTIN CALLEBERG

LINUS HAGVALL

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

Software Architecture for Reducing Testing Effort while Ensuring Safety in Autonomous Road Vehicles

MARTIN CALLEBERG
LINUS HAGVALL

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2017

Software Architecture for Reducing Testing Effort while Ensuring Safety in Autonomous Road Vehicles
MARTIN CALLEBERG
LINUS HAGVALL

© MARTIN CALLEBERG, 2017.
© LINUS HAGVALL, 2017.

Supervisors:
Christian Berger, Computer Science and Engineering
Jan Schröder, Computer Science and Engineering
Leo Laine, Volvo Group Trucks Technology
Fredrik Sandblom, Volvo Group Trucks Technology

Examiner:
Robert Feldt, Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

Software Architecture for Reducing Testing Effort while Ensuring Safety in Autonomous Road Vehicles
MARTIN CALLEBERG
LINUS HAGVALL
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This study proposes a software architecture which separates safety constraints from functionality within the context of decision making in autonomous road vehicles. The intention of the separation is to avoid having to recertify safety requirements when updating functionality. The scope is limited to only solving problems which exist as a result of separating safety constraints and functionality. A literature review was done in order to identify common software architectural patterns used in connection with safety in autonomous road vehicles. The patterns were used to design a general architecture which was evaluated through application in an industrial case.

The resulting system is divided into a decision component and a safety component. The decision component is responsible for passenger comfort and for reaching the desired destination. The safety component is responsible for ensuring safety and has the power to override any action proposed by the decision component. A prototype was implemented and showed promising results. While the proposed architecture does not necessarily make the initial development easier the benefits are realized later in the life cycle through faster and cheaper verification of new functionality.

Keywords: software architecture, autonomous vehicle, safety, testing effort, decision making

Acknowledgements

We are grateful for the opportunity to write our master thesis within a highly relevant and interesting topic at Volvo Group Trucks Technology. We would like to thank our industrial supervisors, Leo Laine and Fredrik Sandblom, and Peter Nilsson for constructive discussions and continuous assistance throughout the project. We would also like to thank Anders Magnusson for his input regarding both our proposed architecture as well as the reference architecture used at Volvo Group Trucks Technology.

Last, but definitely not least, we would like to thank our academic supervisors, Jan Schröder and Christian Berger. An extra thank you to Jan Schröder for his continuous support and constructive comments regarding the thesis.

Martin Calleberg, Gothenburg, June 2017
Linus Hagvall, Gothenburg, June 2017

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
- 1.1 Problem Statement . . . 1
- 1.2 Purpose . . . 2
- 1.3 Research Questions . . . 3
- 1.4 Scope . . . 3

2 Background **5**
- 2.1 Autonomous Road Vehicles . . . 5
- 2.2 Safety Classification . . . 6

3 Method **7**
- 3.1 Literature Review . . . 7
  - 3.1.1 Identification of Research . . . 7
  - 3.1.2 Inclusion Criteria . . . 8
  - 3.1.3 Study Selection . . . 9
  - 3.1.4 Data Extraction and Synthesis . . . 10
- 3.2 General Architecture . . . 10
  - 3.2.1 Requirement Elicitation . . . 10
  - 3.2.2 Design . . . 11
- 3.3 Industrial Case . . . 11
  - 3.3.1 Requirement Elicitation . . . 12
  - 3.3.2 Architecture Adaptation . . . 13
  - 3.3.3 Evaluation . . . 13

4 Literature Review **15**
- 4.1 Layer . . . 17
- 4.2 Monitor and Watchdog . . . 18
- 4.3 Instant Action . . . 20
- 4.4 Faults and Redundancy . . . 21

5 General Architecture **25**
- 5.1 Requirements . . . 25
- 5.2 Responsibilities . . . 26

5.3 Safety Component 28
5.3.1 Safeset Calculator 29
5.3.2 Action Selector 30
5.3.3 Redundancy 31
5.3.4 Monitor 33

6 Industrial Case 35
6.1 Requirements 35
6.2 Safety Component 36
6.2.1 Prediction 39
6.2.2 Path Testing 40
6.2.3 Verification 41
6.2.4 Supported Requirements 42
6.3 Prototype 43
6.3.1 Implementation Scope 43
6.3.2 Scenarios 43
6.4 Evaluation of Results 45

7 Discussion 47
7.1 Outcome 47
7.2 Assumptions 48
7.3 Single Point of Failure 49
7.4 Alternative Applications 50
7.5 Alternative Solutions 51
7.6 Limitations 52

8 Conclusion 55
8.1 Contribution 55
8.2 Future Work 56

Bibliography 57

A Highway Scenarios I

B Industrial Case Requirements III
B.1 Context Requirements III
B.2 Safety Requirements IV
B.3 Performance Requirements V
B.4 Architecture Requirements VI

List of Figures

2.1 Three typical components in an autonomous road vehicle. 5

3.1 The process of the literature review. 9

4.1 Pattern for a layered solution. 17
4.2 Left: pattern for a monitor. Right: pattern for a watchdog. 18
4.3 Pattern for instant action. 20
4.4 Example of redundancy. 21

5.1 Component diagram for the major components in an autonomous vehicle. 26
5.2 Component diagram with the proposed safety component added. . . . 27
5.3 Top level architecture of the safety component. The connections to *Monitor* is left out as it is connected to all other components. 28
5.4 Diagram of the subcomponents inside *SafesetCalculator*. 29
5.5 Diagram of the subcomponents inside *ActionSelector*. 31
5.6 Diagram showing how redundancy could be implemented. The connections to *Monitor* is left out as it is connected to all other components. Also note that not all connections between the instances of *SafesetCalculator* and *ActionSelector* are shown. 32

6.1 The process, within the safety component, of defining safe actions and verifying a proposed action from the decision component. 37
6.2 Abstract class diagram with the most central classes in the safety component. Three classes are written in italics and represent the connection to surrounding components. 38
6.3 The reachable area of a traffic participant has been marked in red. In this case, the reachable area has been simplified to a rectangle and is only used to illustrate the principle. 39
6.4 The reachable area of the traffic participant from Figure 6.3 has been limited to the lane of the traffic participant. 39
6.5 Here a number of tested paths can be seen. Paths which are not considered safe are marked in red and paths which are considered safe are marked in green. 41

A.1 Scenario 1: One lane without roadsides and with no surrounding vehicles. I

A.2 Scenario 2: One lane without roadsides and with a vehicle in front of the ego vehicle . . . I
A.3 Scenario 3: Multiple lanes with a vehicle braking in front of the ego vehicle. . . . I
A.4 Scenario 4: Multiple lanes and no other vehicles. The ego vehicle performs a lane change. . . . I
A.5 Scenario 5: Multiple lanes, the ego vehicle attempts to change into a lane where there are vehicles behind and in front of the ego vehicle. . I
A.6 Scenario 6: Multiple lanes, the ego vehicle attempts to overtake a vehicle in another lane. . . . II
A.7 Scenario 7: Multiple lanes where the traffic in a lane next to the ego vehicle's lane is dense and moves slowly. . . . II
A.8 Scenario 8: Multiple lanes where the ego vehicle is being overtaken by a vehicle in another lane. . . . II
A.9 Scenario 9: Multiple lanes where a vehicle changes into the ego vehicle's lane shortly after overtaking the ego vehicle. . . . II

List of Tables

4.1 Solution used by the relevant papers in the literature review. 16
4.2 Categorization of papers according to level of detail and focus area. . 17

6.1 Summary over results of scenarios. 44

1

Introduction

Safety-critical systems are systems which, when failing, can cause consequences deemed unacceptable, generally referring to accidents involving humans or damage to property or the environment [1]. This puts stricter requirements on different parts of the development process, such as specification, architecture and verification of the system.

Systems for autonomous road vehicles can be considered safety-critical as they have the potential to cause accidents with both material damage and human injury [2]. Autonomous vehicles can be autonomous to different degrees, ranging from adaptive cruise control to driving completely autonomous without the need for a standby driver [3]. Vehicles with a higher autonomous degree have more and stricter safety requirements.

1.1 Problem Statement

Developing a system which is hard to change is typically uneconomical as the requirements for a system tend to change during its lifetime [4]. When the functionality is changed to meet new requirements, the system needs to be tested to verify that it performs as expected [5]. Furthermore, to ensure old functionality still works, everything which could have been affected by the change also needs to be tested by doing regression testing.

Safety-critical systems often need extensive testing to receive a certificate that they are proven, to some degree of uncertainty, to be safe for production [6]. The tests for these types of systems thus need to cover as many scenarios as possible and can be both time consuming and costly to run. One example of safety-critical software is software within vehicles [7], especially autonomous vehicles are expected to require extensive testing [8]. The testing and verification required to put new software in production, or updating old software, creates a sizeable obstacle for providing customers with the latest technology.

Nord et al. [9] argue that this type of problem can be considered an architectural problem. If functionality is split into different architectural components with no direct or indirect dependencies, there is no need to test both functionality areas if only one area is changed. Kelly [10] argues in a similar direction, but instead of arguing for different components he uses a logical division of safety-critical aspects into different safety cases. Both approaches build upon a principle of dividing functionality and thus limiting the number of dependencies between features. However, this can be problematic in certain contexts. In the context of autonomous machines Behere et al. [11] argue that there is a tendency towards requiring increased communication between components.

Accordingly, combining the thorough testing required for safety-critical systems with the ability to provide frequent updates is a challenge. There are some approaches which could reduce the need for testing by moving safety aspects to separate components. Underwood et al. [12] present an approach where they focus on autonomous trucks. They suggest a safety layer which verifies all actions before they are carried out by the vehicle. Underwood et al. speculate that their architecture would only require safety certification of the safety layer and certain vehicle safety modules in other layers.

Another approach is used by Liu and Özgüner [13] who use a separate communication path for emergency scenarios. Based on the perceived environment a decision is made for whether an emergency action by a safety component is required or if normal operation can continue.

A third approach is presented by Behere and Törngren [14]. They use a reactive control component which has the ability to override any actions taken by other components. The reactive control component runs on a higher frequency and triggers an action when an immediate threat is detected.

A common factor for the mentioned studies is that they discuss the task and responsibilities of their safety components but do not go into detail about how it could be implemented. Behere and Törngren [14] specifically mention that there is a need for more research regarding details of the different components.

These studies all propose solutions which could be used to reduce testing of autonomous road vehicles through decoupling safety aspects from decision making. However, there still remains a need for studies which focus on how such a safety component could be designed.

1.2 Purpose

The purpose of the study is to propose a software architecture which separates safety constraints from functionality within the context of autonomous road vehicles. The intention of the separation is to avoid having to recertify the implementation of safety constraints in case of functionality updates.

1.3 Research Questions

In order to fulfill the purpose, of proposing a software architecture for autonomous road vehicles which separates safety constraints from functionality, the following research questions are investigated:

RQ1 Which software architectural patterns and tactics are used in literature to increase safety in autonomous road vehicles?

RQ2 How can the patterns and tactics from RQ1 be used to design a general architecture which separates safety constraints from functionality?

1.4 Scope

The proposed architecture focuses on how to separate safety constraints from functionality in an autonomous road vehicle. More precise, the software components controlling the autonomous road vehicle are considered. The study only focuses on safety constraints related to the driving activity. While it could be argued that there are a number of different safety related activities, such as the unloading and loading of a vehicle, these are not included in the study.

The study does not present ideas on how to implement functionality needed for an autonomous road vehicle to operate. The study only attempts to solve problems which are created as a result of the separation of safety constraints and functionality. As a result, problems which need to be solved regardless of whether the decision component is separated or not is not considered. Accordingly, the study does not validate safety constraints and assumes that all requirements are correctly specified. However, the division of responsibilities between functionality and safety is clearly defined.

This study focuses on the part of a system which handles short term decision making with a time horizon of up to a few seconds. Furthermore, the decision making should be able to make decisions without external help. This includes that the vehicle is not allowed to rely on a backup driver or be dependent on communication with other vehicles or infrastructure systems.

Safety related problems in areas separated from decision making are not considered. All incoming sensor data, both regarding the vehicle status and the surrounding environment, is assumed to be correct. The same thing goes for the actuators which control the vehicle. Although the architecture is from a software perspective, it is still necessary to design an architecture which allows for hardware replication.

2

Background

In this chapter concepts related to autonomous road vehicles are presented. The typical major components in an autonomous vehicle are described as well as how safety requirements for road vehicles can be classified.

2.1 Autonomous Road Vehicles

All autonomous road vehicles typically have three major components [15], which can be seen in Figure 2.1. The perception component is responsible for gathering information about the surrounding environment. To do this the component generally uses a set of sensors mounted on the vehicle, such as different types of cameras and Global Positioning System (GPS) antennas. Using sensors the perception component can detect road surface and where other traffic participants and obstacles are in relation to the ego vehicle. The ego vehicle refers to the vehicle the system runs on.

The decision component is responsible for piloting the vehicle [15] and corresponds to the human driver in traditional vehicles. The component can consist of several types of guidance systems making plans for different time scales. Some systems might be handling the overall route, and thus focus on which intersection to turn at, while other systems decide the trajectory for the next few seconds.

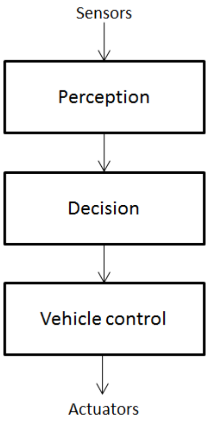


Figure 2.1: Three typical components in an autonomous road vehicle.

The final component, the vehicle control component, is responsible for executing requests generated by the decision component [15]. The vehicle control component calculates which actuators to use in order to fulfill the requests from the decision component.

2.2 Safety Classification

Safety requirements in the automotive industry can be classified with an Automotive Safety Integrity Level (ASIL), defined by ISO 26262 [16]. These levels are determined by performing a risk analysis of the system for potential hazards. From the result of the analysis different requirements for the system are given an ASIL classification between A and D, where D is the highest level and thus denoting a requirement which has high criticality in regards to safety. However, if a requirement is not safety-critical it is classified as ASIL QM.

When performing the risk analysis the severity, exposure and controllability in the event of a certain fault is considered. Severity refers to the severity of the accidents which could occur as a result from the fault, exposure refers to how often the vehicle is in a situation where the fault would be dangerous and controllability refers to the ability a driver has to mitigate the fault. Each parameter is classified with a value according to a set of guidelines. The appropriate ASIL classification is given by a lookup table using the three parameters.

ASIL requirements can be decomposed into two requirements with lower ASIL classification. The sum of the two resulting ASIL classifications has to be greater or equal to the old ASIL classification, for example ASIL D can be decomposed into ASIL B + ASIL B or ASIL A + ASIL C. However, the decomposition can only be done if the resulting requirements are completely independent.

3

Method

This study can be divided into three different phases: literature review, general architecture and industrial case. The first phase, literature review, relates to RQ1 and creates a foundation for the rest of the study to build on. The general architecture and the industrial case relates to RQ2, where the industrial case functions as an evaluation of the general architecture.

3.1 Literature Review

The first part of the study was a literature review focusing on architectural patterns and tactics for safety in autonomous road vehicles. The literature review was conducted using a process adapted from Kitchenham's [17] methodology. The review extracted data both regarding which patterns and tactics are used as well as their described characteristics. The result of the literature review answers RQ1, to find which software architectural patterns and tactics are currently used to increase safety in autonomous road vehicles.

3.1.1 Identification of Research

To find relevant research papers a suitable search query had to be generated. Initially, trial searches were performed and the results analyzed for relevance and keywords. Synonyms for the identified keywords were then added to avoid excluding potentially relevant research papers.

Using the identified keywords the search query was divided into three parts. The first part aimed to limit the result to the relevant application area of autonomous road vehicles:

(autonomous OR automated) AND (automotive OR car OR truck OR bus OR road OR drive OR driving)

The second part of the query aimed to limit the result to the relevant technology area:

(software OR system) AND architecture

The last part of the query aimed to limit the result to the desired focus of improving safety in autonomous road vehicles:

(safety OR safe)

The complete search query was used on four scientific databases. The selected databases for this study were ACM Digital Library[1], IEEE Xplore Digital Library[2], ScienceDirect[3] and Web of Science[4]. The searches were done on the title, abstract and keywords of research papers, which resulted in 78, 369, 36 and 226 papers respectively. Beside the databases, there were 18 additional papers suggested by field experts.

3.1.2 Inclusion Criteria

RQ1 and the purpose led to a couple of inclusion criteria which had to be fulfilled. Since the study focuses on autonomous road vehicles, the same had to be true for research papers in the literature review. Even if a paper did not focus on autonomous road vehicles, it could be included if it explicitly stated it as an area where the result could be applicable.

As RQ1 is in the context of software architecture, the literature review had the same focus. Studies were included if they provided solutions within, or partly within, the context of software architecture. Thus papers which discussed an architecture which combines hardware and software architecture were included. In order to be included the paper also had to provide some sort of solution or approach to solve a problem. The literature review did not include papers which only brought attention to problems.

Given the scope of the study, see section *1.4 Scope*, included papers had to discuss safety in the context of decision making. This resulted in two inclusion criteria: in order to be included a paper had to discuss an architecture which includes at least one safety aspect and be at least partly within the context of decision making.

The scope also limits the study to autonomous vehicles that are able to operate without external help. This means that the solution provided by a paper should not be dependent on a standby driver nor on communication with other vehicles or traffic infrastructure systems.

This can be summarized in six inclusion criteria:

- The paper must focus on autonomous road vehicles or use autonomous road vehicles as an example where their result could be applicable.
- The paper must discuss an architecture with software aspects.

---

[1] http://dl.acm.org/

[2] http://ieeexplore.ieee.org

[3] http://www.sciencedirect.com/

[4] https://webofknowledge.com/

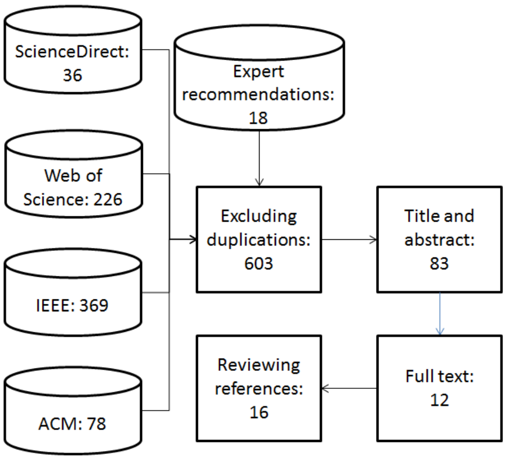


Figure 3.1: The process of the literature review.

- The paper must provide a solution or approach, as opposed to only bringing attention to an issue.
- The paper must discuss an architecture which contains a safety aspect.
- The paper must discuss an architecture which contains some information regarding decision making.
- The paper must discuss an architecture where the vehicle is not dependent on the driver, communication with other vehicles or communication with infrastructure systems.

3.1.3 Study Selection

An overview of the study selection process can be seen in Figure 3.1. Initially, the title and abstract of all papers were studied for whether they should be further investigated. As suggested by Kitchenham [17], the inclusion criteria were interpreted liberally when reading the title and abstract of a paper. No paper was excluded unless it was clear that it did not meet the inclusion criteria. All papers were reviewed by both researchers and no paper was excluded unless both researchers agreed. After removing duplicates and excluding non-relevant papers the number of papers was reduced to 83.

The researchers divided the remaining papers and read through them in full with the same inclusion criteria as previously in mind. However, the inclusion criteria were not interpreted as liberally as before. At this point papers were excluded even if only one researcher had read it through. If it was unclear whether a paper should be excluded it was discussed together with the other researcher. After excluding papers based on their full text, 12 papers remained.

The 12 papers with relevant content were used to find additional research. Kitchenham [17] suggests that the reference lists of relevant research papers are searched for more potentially relevant papers. By examining the references 4 additional papers were deemed to be of interest, increasing the total number of papers to 16.

3.1.4 Data Extraction and Synthesis

The goal of the literature review was to identify all proposed safety related solutions as well as the problems the papers were trying to solve. If a paper provided a discussion of their proposed solution it was examined for thoughts on the solution's characteristics. This study considers the following characteristics interesting: drawbacks, advantages and rationale for choosing the solution. The papers were divided between the researchers and carefully read through. All parts which the assigned researcher deemed as potential candidates for extraction were discussed between both researchers and a common decision was made for what information passed the extraction criteria.

The proposed solutions were compared for similarities and grouped accordingly. Within each group the solutions were compared for differences, both regarding the actual solutions and corresponding characteristics. The result was a list of common patterns studied in relation to autonomous road vehicles regarding safety. This list was used to answer RQ1, to find which software architectural patterns and tactics are currently used to increase safety in autonomous road vehicles.

3.2 General Architecture

The result of the literature review was used to develop a general architecture. The goal of the architecture was to separate safety constraints from functionality and to provide a base for answering RQ2. This section presents the method for developing the architecture.

3.2.1 Requirement Elicitation

In order to develop an architecture it is necessary to have requirements for what it must support. While the general architecture in this study should avoid case specific requirements there still are some constraints the architecture needs to adhere to. As stated in RQ2, the architecture should be general and thus not provide any implementation specific details. This results in that the requirements should stay on the same abstraction level.

The requirements were elicited from three sources: the context of the study, the literature review and the industrial case. The purpose and scope defines a context from which the majority of the requirements could be identified. These requirements were also corroborated by the selected papers in the literature review. As the selected papers in the literature review studied autonomous road vehicles, the requirements presented for the studied systems could potentially be applied in this study as well. However, requirements which were specific to the cases described in the papers were ignored. The general requirements were later complemented with context independent requirements which surfaced during the elicitation for the industrial case. The requirement elicitation resulted in a list of requirements which was used when designing the general architecture.

3.2.2 Design

The general architecture was developed using Attribute-driven design (ADD)[18]. ADD is a process where requirements which can have an impact on the architecture are identified as architectural drivers. The architectural drivers are later used to iteratively develop a system architecture. However, the first step is to identify all requirements, see section *3.2.1 Requirement Elicitation*.

The second step, using ADD, was to select a part of the system to decompose into smaller components. The initial system considered for decomposition was the typical architecture described in section *2.1 Autonomous Road Vehicles*. Due to the purpose and scope of the study, only the decision component was further decomposed into an increasingly more detailed architecture. For every decomposition the requirements were checked for whether they were architectural drivers for any part of the system.

In accordance with ADD, the next steps were to evaluate and apply different architectural patterns to the system. In an iterative fashion, an unfulfilled architectural driver was selected to be considered in the next decomposition of the system. Different patterns were assessed for how well the current architectural driver could be fulfilled using the pattern, where the most promising pattern was applied to the architecture. In some cases small alterations were necessary for the pattern to fit.

At the end of an iteration the final step was to verify that the architecture fulfilled the intended architectural drivers. This included both the architectural drivers for the current iteration as well as from previous iterations. The process finished when the architecture provided support for all architectural drivers and all parts were decomposed into sufficient detail.

3.3 Industrial Case

The preliminary general architecture, or in other words the preliminary answer to RQ2, was evaluated through application in an industrial case using action research. Action research involves five steps: diagnosing, action planning, action taking, evaluating and specifying learning [19]. The process is iterative and earlier steps are revisited as problems are discovered or new knowledge is acquired. Action research includes close collaboration with practitioners, both when it comes to designing a solution and evaluating the result.

The action research was done in collaboration with Volvo Group Trucks Technology. Through this collaboration the researchers gained access to a system under development for an autonomous long vehicle combination. The company's goal was to be able to update the decision making component without affecting safety. This provided an opportunity to apply and test the general architecture in practice. The architecture used at the company is similar to the typical architecture for autonomous road vehicles described in section *2.1 Autonomous Road Vehicles*.

Four senior industry experts at Volvo were used to provide insight in fields which were necessary for the study. One industrial expert is within active safety, one within software architecture and two within vehicle dynamics. These fields contain information regarding: how to keep a vehicle safe, software architecture within vehicles and how a vehicle can behave. One expert from each field have leading technical positions and have had exposure towards vehicle automation. The fourth expert, within vehicle dynamics, is currently finishing his Ph.D. related to automation of long vehicle combinations and have written parts of the decision component studied in the industrial case.

The process of applying the general architecture is divided into three different parts. The following sections describe the requirement elicitation, the architecture adaptation and the evaluation process.

3.3.1 Requirement Elicitation

The first step in action research, diagnosing, is to identify problems. In this case the general problem of the study was already defined and was also present in the industrial case. In order to be able to address the details of the industrial case it was necessary to elicit case specific requirements.

Initially, four unstructured interviews were carried out with the previously mentioned industry experts. The purpose of an unstructured interview generally is to gather as much information as possible within a loosely defined topic [20]. The interviews had open-ended questions to allow the experts to freely discuss the topic, including both what they considered part of the problem as well as which aspects they considered most important. The result from the interviews was used to get an overview of the problem and determine which areas to focus on initially.

The existing system was studied through internal documentation. When studying the documentation, communication between relevant system components were investigated for what information they provided and required. Among other things, this included the communication between the perception, decision and vehicle control components.

The information from both the unstructured interviews and the studied documentation was used in eight workshops and three additional interviews. The interviews were semi-structured and conducted with the same field experts as in the initial interviews. Although the discussions during the workshops and interviews were open-ended, they started with an existing problem in focus. Since the process was iterative, a workshop or interview was often used both to confirm and reiterate requirements as well as to develop and evaluate different potential solutions.

3.3.2 Architecture Adaptation

Applying the general architecture to the system provided by Volvo meant it was necessary to identify areas in need of adjustments and to develop a more detailed architecture. The first step was to identify changes between the architecturally significant requirements in the Volvo case and the general case. The Volvo requirements were often more detailed, or context specific, versions of the requirements for the general architecture.

The second step was to study the system's current architecture in order to identify where the solution could be applied. This meant identifying the responsibilities of the different components within Volvo's system. The most important part was to identify where different safety constraints were currently handled.

The next step was to plan how the general architecture could be applied. The planning included both decomposing the architecture and looking at whether changes were required. This was done using ADD [18], similarly to how the general architecture was initially created (see section *3.2.2 Design*).

The final step was to define the communication between the components, with extra effort on deciding what information the solution requires and provides. This was done in order to be able to determine the requirements and constraints the solution would put on the rest of the system.

As mentioned in 3.3.1 Requirement Elicitation, there were continuous workshops with industry experts which focused both on evaluating different approaches and on solving problems which had surfaced. Advantages and disadvantages between different approaches were compared before eventually being chosen or rejected.

3.3.3 Evaluation

As previously mentioned in section *3.3.1 Requirement Elicitation*, the industry experts evaluated the solution continuously throughout the project during workshops. The focus of the workshops differed depending on the attending industry experts, for example the active safety expert focuses his expertise on the safety related aspects of the project. The result was an iterative approach with an evolving architecture. As new ideas emerged in the industrial case they were also evaluated for applicability in a general context, in other words being applicable in the general architecture. The evaluation of the architecture mainly focused on its ability to handle different kinds of problems. The problems differed between being well defined to only being an abstract idea of what could become a problem in the future.

An implementation of the architecture was developed as a proof of concept prototype to show whether the architecture was feasible to implement. The prototype is only an implementation of the safety component and uses input from already existing systems. The prototype is limited to handling a set of highway scenarios. Each scenario was tested to see whether the solution would be to able to meet both functional and non-functional (quality) requirements. The scenarios were tested in an already existing simulator and test that the vehicle never leaves a safe state and how the vehicle is allowed to operate. The results from the tests were used in workshops to discuss the design choices and possible improvements as well as to refine the architecture for RQ2.

4

Literature Review

The literature review for software architectural patterns and tactics for safety in autonomous road vehicles resulted in 16 relevant papers. The solutions provided by the papers were divided into five different categories and can be seen in Table 4.1.

The papers were also categorized according to their focus area and level of detail, see Table 4.2. While some papers discuss the solutions in great detail, other papers discuss it at a conceptual level. Furthermore, the papers have different focus were some discuss safety in more detail than functionality while some discuss functionality in more detail than safety. There were also papers with a more general focus. However, regardless of focus, all papers provide information regarding safety aspects in accordance with the inclusion criteria, see section *3.1.2 Inclusion Criteria.* The 16 included papers and their characteristics are described in the following sections and answers RQ1, of which software architectural patterns and tactics are used in literature to increase safety in autonomous road vehicles.

Several of the papers use different vocabularies when discussing the same subject. In order to be able to compare the papers more easily a shared vocabulary is used when presenting them, for example calling similar components by the same name. When applicable the same naming as described in section *2.1 Autonomous Road Vehicles* for autonomous road vehicles is used.

Table 4.1: Solution used by the relevant papers in the literature review.

ID	Title	Layer	Monitor	Watchdog	Instant action	Faults and redundancy
A	Using Dynamic Adaptive Systems in Safety-Critical Domains [21]					X
B	Human Driver Model and Driver Decision Making for Intersection Driving [13]				X	
C	Safe and Efficient Runtime Resource Management in Heterogeneous Systems for Automated Driving [22]		X			X
D	Strategy and architecture of a safety concept for fully automatic and autonomous driving assistance systems [23]		X			
E	Towards trustworthy intelligence on the road: A flexible architecture for safe, adaptive, autonomous applications [24]				X	
F	Comparison of fail-operational software architectures from the viewpoint of an automotive application [25]					X
G	Truck Automation: Testing and Trusting the Virtual Driver [12]	X				X
H	SAFER: System-level Architecture for Failure Evasion in Real-time Applications [26]			X		X
I	A functional reference architecture for autonomous driving [14]				X	
J	Caroline: An autonomously driving vehicle for urban environments [27]			X		
K	The future of driving, Deliverable D21.1 Architecture [28]				X	
L	A hardware and software framework for cognitive automobiles [29]			X		
M	Design and capabilities of the Munich Cognitive Automobile [30]	X				
N	Design of the planner of team AnnieWAY's autonomous vehicle used in the DARPA Urban Challenge 2007 [31]	X				
O	Junior: The Stanford Entry in the Urban Challenge [32]			X		
P	Making Bertha Drive—An Autonomous Journey on a Historic Route [33]				X	

Table 4.2: Categorization of papers according to level of detail and focus area.

Focus \ Detail	Conceptual solution	Detailed solution
Safety	D, G	A, E, F, H
General	C, I, K, L, M, P	J
Functionality	N, O	B

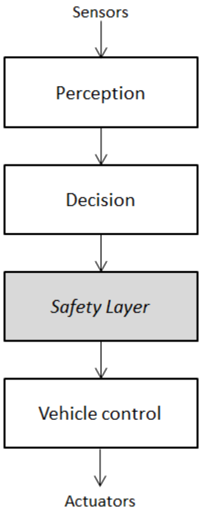


Figure 4.1: Pattern for a layered solution.

4.1 Layer

Underwood et al. [12] (G), Goebl et al. [30] (M) and Gindele et al. [31] (N) describe architectures where trajectories are verified before they are executed. All three approaches are considered conceptual solutions, see Table 4.2, and build upon the same principles. Instead of sending the planned trajectory directly to the vehicle control component the trajectory is sent through a safety layer which verifies it. If the layer deems the trajectory unsafe it can modify or reject the trajectory before forwarding it to vehicle control. In Figure 4.1 an abstract software pattern for this safety layer can be seen. The layer's goal of increasing safety results in a constraint that all communication to the vehicle control component must go through the layer. If another component can communicate directly with the vehicle control component it will negate the safety layer's ability to ensure different safety aspects.

Gindele et al. [31] (N) are sparse with additional details regarding the safety layer. However, Underwood et al. [12] (G) and Goebl et al. [30] (M) provide more insight and discussions. A safety layer as described by Underwood et al. checks if a trajectory will lead to collisions and modifies it to avoid accidents. If the component generating trajectories fails the safety layer will take control of the vehicle and park it in a safe location. The rationale given by Underwood et al. for using a layer is

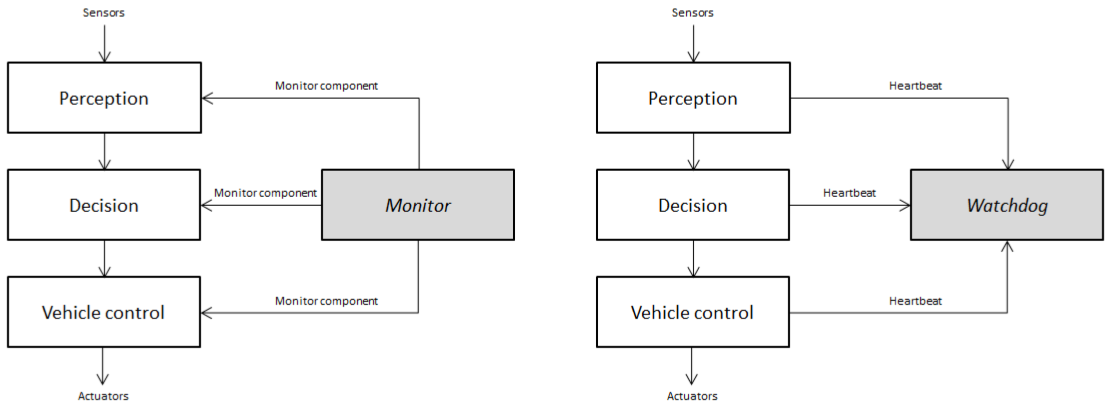


Figure 4.2: Left: pattern for a monitor. Right: pattern for a watchdog.

to ease the safety verification of the system. This is achieved by decomposing the system by functionality and thus separating safety aspects into a separate layer. Underwood et al. speculate that it could be sufficient for safety certification to prove that the system can detect faults and keep the vehicle in a safe state. However, Underwood et al. do not provide details on how it can be realized.

As opposed to Underwood et al. [12] (G), Goebl et al. [30] (M) do not discuss rationale and intent for using a safety layer. Instead, Goebl et al. focus on how the safety layer can ensure safety of a trajectory. By calculating the set of all possible positions a traffic participant can reach, a trajectory can be checked for potential collisions. If a trajectory does not intersect with any set it can be guaranteed as safe. Contrary to Underwood et al. [12] (G) and Gindele et al. [31] (N), Goebl et al. do not specify how the layer should respond to unsafe trajectories.

4.2 Monitor and Watchdog

An approach commonly used among the selected articles was to use a monitor or watchdog to ensure that different parts of the system run as intended. Even though the level of detail and focus area varies, see Table 4.2, there are still similarities in that a dedicated part of the system is responsible for performing an emergency action if there is a critical system failure.

The watchdog approach is used in different forms by Rauskolb et al. [27] (J), Werling et al. [29] (L), Kim et al. [26] (H) and Montemerlo et al. [32] (O). These papers present solutions where one or more watchdog components listen to output of other components, a simple variant of a watchdog architecture is shown in Figure 4.2. Depending on the solution a watchdog can listen to heartbeats directly from different components or listen to their communication with other components. The responsibility of a watchdog can also vary depending on both the watchdog's problem solving capabilities and on whether the watchdog is distributed or centralized.

Both Rauskolb et al. [27] (J) and Montemerlo et al. [32] (O) describe an approach where watchdogs listen to heartbeats from different processes throughout the system.

The watchdogs can restart processes and, if deemed necessary, perform an emergency stop of the vehicle. The concept is taken a bit further by Rauskolb et al. who use one master watchdog and several slave watchdogs. Different components have their own local slave watchdog sending data to a central master watchdog which can perform an emergency stop if necessary. Rauskolb et al. argue that the distributed watchdogs with their ability to restart different processes independently maximize the self-healing capability.

Werling et al. [29] (L) have a different approach and propose a framework with a central real-time database which different components read from and write to. This allows them to use a data watchdog which only looks at the data inside the database. If a process does not input data for a certain amount of time it will be restarted. In case the process still does not input data an emergency stop of the vehicle is performed. To detect if the data watchdog or database fails, there is another watchdog which only checks that the data watchdog and database are running.

Kim et al. [26] (H) turn around completely and do not have a dedicated watchdog component. Kim et al. discuss an approach where it is necessary to run several identical software processes. The proposed solution is to have every process send a heartbeat to other identical processes. This means that different processes watch each other and can take over if a process fails.

Borrman et al. [22] (C) and Hörwick and Siedersberger [23] (D) use a monitoring approach. A monitor component differs from a watchdog component in that it actively requests information as opposed to passively listening, see Figure 4.2. Hörwick and Siedersberger present an architecture where they use both a local monitor in each larger block as well as a global monitor. The approach can be compared with Rauskolb et al. [27] (J) who use a similar distributed structure but with watchdogs. Hörwick and Siedersberger mention that some faults can be discovered through a final plausibility check by the global monitor while some faults only can be discovered locally. For example it is hard to discover faults from perception in a global monitor.

Borrman et al. [22] (C) describe a solution where the monitor has different responsibilities from previously discussed papers. The monitor determines the state in which the system runs based on the execution time of certain processes. By deciding a state the monitor determines which actions can be taken by the vehicle. Examples of this could be how hard the vehicle is allowed to brake and how sharp it is allowed to turn. In case of severe system failure the monitor can select a safe error state which can perform an emergency stop.

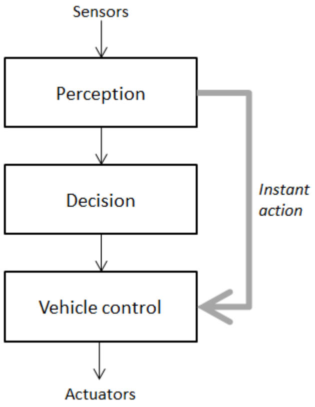


Figure 4.3: Pattern for instant action.

4.3 Instant Action

Liu and Özgüner [13] (B), Behere and Törngren [14] (I), Jakobsson et al. [28] (K) and Ziegler et al. [33] (P) all describe architectures where components can be bypassed if an emergency maneuver is deemed necessary. When compiling data from the vehicle's sensors it can be discovered that there is an obstacle dangerously close. Instead of sending the sensor data to a decision component the system could send a message directly, or through a dedicated emergency component, to the vehicle control component. An instant emergency action could then be performed without having to wait for the decision component. Figure 4.3 shows an abstract software pattern for this type of behavior.

Jakobsson et al. [28] (K) present an architecture with a decision component which decides a destination for the vehicle and a separate component which generates the trajectory. If the perception component considers it necessary it can bypass the decision component and directly communicate with the component generating trajectories. Ziegler et al. [33] (P) present a system using the same approach as Jakobsson et al. However, Ziegler et al. suggest that the instant action can be implemented by the vehicle's active safety functions. As such safety features can remain active when vehicles become autonomous, the autonomous system does not have to include an implementation of the same functionality. Behere and Törngren [14] (I) and Underwood et al. [12] (G) present similar arguments that some safety can be given by the collision avoidance already present in some modern cars.

The solution described by Liu and Özgüner [13] (B) use a dedicated component to handle emergencies. This component is triggered when the perception component detects an obstacle dangerously close or if other vehicles break traffic rules. When triggered, it will take control of the vehicle by overriding the requests sent from the decision component and try to put the vehicle in a safe state. As seen in Table 4.2, Liu and Özgüner do not focus on safety and thus do not provide more relevant details.

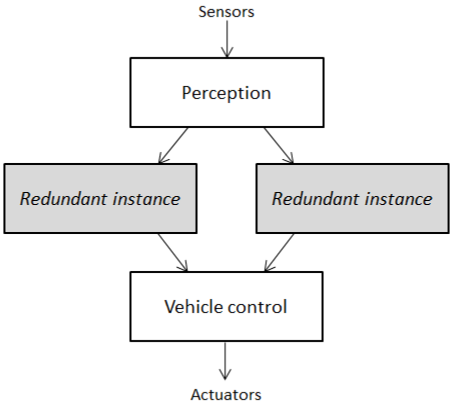


Figure 4.4: Example of redundancy.

Similarly to Liu and Özgüner [13] (B), Behere and Törngren [14] (I) describe an architecture which has a dedicated component for handling emergencies. The dedicated component is instantly triggered when something unanticipated is detected by the perception component. However, Behere and Törngren note that if the vehicle's decision component is sufficiently fast, it could detect the threat and make an appropriate decision similar to that of the dedicated emergency component. In that case the dedicated component would only be used as a redundancy measure.

Brancovici [24] (E) presents a similar approach to Liu and Özgüner [13] (B) and Behere and Törngren [14] (I), but with a component between the perception and decision components. The proposed component can decide whether to communicate to the decision or the vehicle control component depending on what the perception component reports. If there is a risk for a potential accident the component will communicate directly with the vehicle control component, otherwise it will address the decision component for more intelligent decisions.

4.4 Faults and Redundancy

A common topic among the selected articles was how to deal with faults and create a fail-operational system. Faults considered are if a component stops responding or if it gives the wrong output, both permanently and transiently. In the selected papers, regardless of whether it relates to software or hardware faults, the solution usually includes some variation of redundancy [21] [22] [26] [25] [12]. An example of redundancy can be seen in Figure 4.4. Creating a fail-operational system includes both addressing how the system can continue to run in the presence of internal faults and how to prevent internal faults from resulting in external faults. As fail-operational systems are generally connected to safety-critical systems it is not necessarily surprising that four out of the five papers related to faults and redundancy have a safety focus, see Table 4.2.

Underwood et al. [12] (G) argue that active safety systems, for example automatic braking before obstacles, should be developed in parallel with autonomous driving systems. The reason for this is that the active safety component can provide redundancy and act as a safety-critical backup to higher levels of automation. They also argue that the active safety component should be able to execute a safe stop if the autonomous system fails. This can be compared to the approach discussed by Borrman et al. [22] (C) who present an approach where a vehicle can be in different states, one of them being a failure state which occurs when one or more systems does not function properly. When in the failure state the vehicle should be able to run a dedicated error-handling process which is able to safely stop the vehicle.

Schnellbach et al. [25] (F) discuss different approaches on how to handle faults from a combined software and hardware perspective. The two approaches which Schnellbach et al. rate highest in terms of fault tolerance capability both involve hardware redundancy as well as how to use the hardware from a software perspective. One approach uses triple redundancy and decides the final output through a voting mechanism. As long as all three modules agree the system stays in a normal state, however, if one module disagrees the system enters a fault tolerant state. At this point Schnellbach et al. recommend that action is taken to restore the functionality of the failing module, this could range from simply restarting the process to visiting a repair shop. If all three modules provide different results the system enters a failure state since the correct decision is unknown.

The second approach proposed by Schnellbach et al. [25] (F) can be used if it is possible to check the result for correctness. The approach involves double redundancy and is quite similar to the previous approach. Since it is possible to determine whether a result is correct or not it is also possible to keep running the system when one of the modules fails and the output differs. This describes a fault tolerant state, but action should be taken to restore the functionality of the failing module.

Kim et al. [26] (H) propose a solution where hot and cold standbys are used as a more flexible solution than hardware replication. Instead of replicating the whole system or subsystem, selected processes can be duplicated onto different CPUs. For processes which are safety-critical, hot standbys can be used which are ready to take over at any time. A hot standby runs on the exact same input but on a different CPU. Only the master process sends its output to other components, but if the master process fails a hot standby is already running and can start providing output. A cold standby is recommended for when time is not as important. A cold standby does not run until it is told, this means that there will be a delay when the process starts up before it is ready to provide output, but on the other hand it will not consume any CPU resources while dormant. However, similarly to Schnellbach et al. [25] (F), this requires a way to decide whether the output is correct.

McGee and McGregor [21] (A) attempt to solve a different problem and provide an approach on how to recover from faults. They suggest that when a process fails, it should be replaced by two new processes. One process should provide basic functionality, enough to meet the system's real-time safety requirements. The other process tries to reconfigure the system, for example through a rollback. The advantage with this solution is that it is possible to create a new process quickly and minimize the risk for breaking safety requirements. However, McGee and McGregor also point out that there is a risk that the process will fail again if the previous system state caused the failure in the first place. Because of this risk, McGee and McGregor advocate for including alternatives to rollback.

5

General Architecture

This chapter presents a general architecture for separating safety constraints from functionality. First the requirements are presented followed by a description of the responsibilities of different system components. Finally the architecture is presented together with descriptions of different components and what tasks they are supposed to perform.

This chapter provides the answer to RQ2, to propose a general architecture which separates safety constraints and functionality in the context of autonomous road vehicles. However, it should be considered that the results presented in this approach was developed through an iterative approach using the industrial case, see chapter *6 Industrial Case*, for continuous evaluation.

5.1 Requirements

A set of requirements was specified for the general architecture as described in section *3.2.1 Requirement Elicitation*. The requirements were used when designing the general architecture and are as follows:

GR1 Separate safety constraints from functionality - The safety constraints should be independent of the decision making of the vehicle. The responsibility for the safety of the vehicle should not be on the decision making component in the system. In other words it should be possible to alter the decision making component without affecting the safety of the vehicle.

GR2 Ensure safety of the vehicle - The system should be able to ensure safe behavior of the vehicle. The vehicle should not be the cause of any accident and, whenever possible, avoid other accidents as well.

GR3 Real-time execution - The system will be executed in real-time and should be able to ensure the safety of vehicle decisions within an acceptable time frame.

GR4 Fail-operational - The system should be able to safely operate in case of system fault.

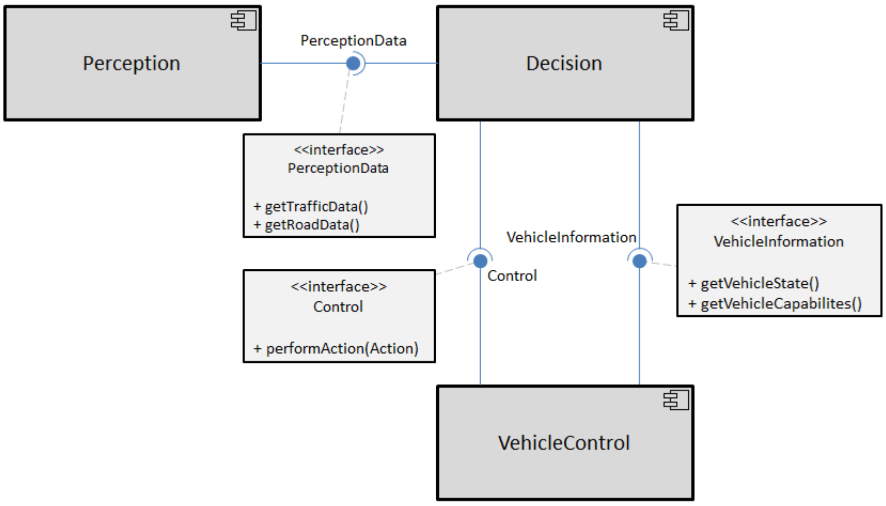

Perception
PerceptionData
Decision
<<interface>>
PerceptionData
+ getTrafficData()
+ getRoadData()
VehicleInformation
Control
<<interface>>
VehicleInformation
+ getVehicleState()
+ getVehicleCapabilites()
<<interface>>
Control
+ performAction(Action)
VehicleControl

Figure 5.1: Component diagram for the major components in an autonomous vehicle.

GR5 Support varying vehicle behavior - The system should be able to handle different rules on how the vehicle is allowed to make decisions.

GR6 Vehicle independent - The system should be able run on different vehicle types.

GR7 Handle different traffic situations - The system should be able to handle different road and traffic contexts.

In the study the requirements were limited in accordance with the scope. In effect, the requirements were limited to the separation of safety constraints and functionality of the decision making component within a software context. More details regarding the scope can be found in section *1.4 Scope*.

5.2 Responsibilities

As specified by requirement GR1 safety should be separated from functionality. This results in one component managing the functionality and one component ensuring the safety. In the context of decision making in an autonomous road vehicle this means the decision component in Figure 5.1 is split into two different components. The safety component will be responsible for all safety aspects but will have no responsibility that the vehicle reaches the destination or to ensure the comfort of passengers. In contrast, the other component, which will still be referred to as the decision component, does not have any formal requirements regarding safety and is responsible for reaching the destination and passenger comfort. Without safety requirements on the decision component it should not be necessary to conduct safety testing before a new version can be used in production. Depending on the vehicle manufacturer the line between safety and functionality could be drawn at different places. For example, some manufacturers might consider that the safety component should enforce traffic rules while other manufacturers might be satisfied if the component only avoids collisions.

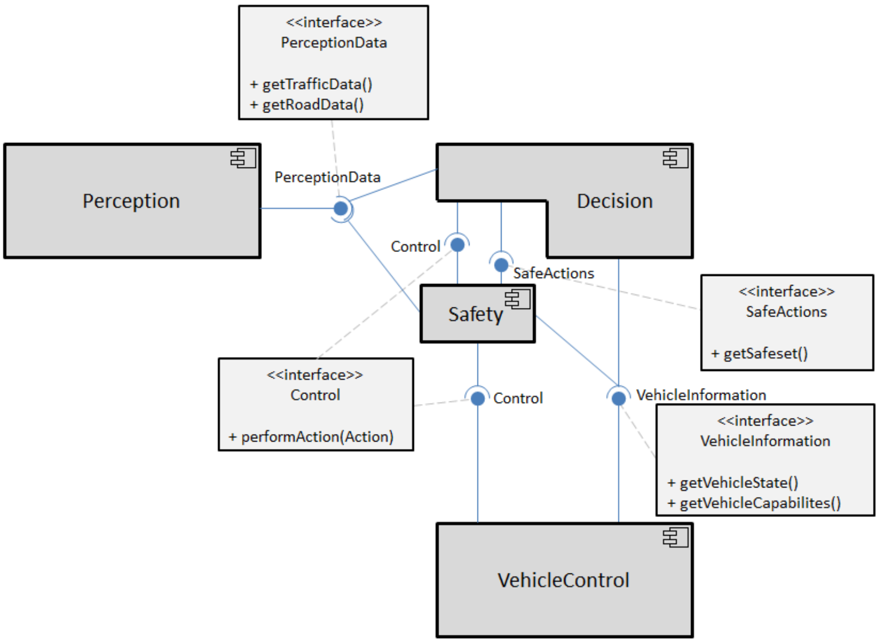

<<interface>>
PerceptionData
+ getTrafficData()
+ getRoadData()
Perception
PerceptionData
Decision
Control
SafeActions
<<interface>>
SafeActions
+ getSafeset()
Safety
<<interface>>
Control
+ performAction(Action)
Control
VehicleInformation
<<interface>>
VehicleInformation
+ getVehicleState()
+ getVehicleCapabilites()
VehicleControl

Figure 5.2: Component diagram with the proposed safety component added.

In order for the safety component to be able to ensure safety on its own it needs to be able to control the vehicle at all times. If the safety component cannot control the vehicle as it desires it cannot meet requirement GR2 of ensuring the safety of the vehicle. This is solved by forcing all control requests for vehicle control to go through the safety component, see Figure 5.2.

Figure 5.2 shows an approach similar to the layer pattern described in section *4.1 Layer*. The difference here is that the safety component, or layer, does not completely hide vehicle control as the vehicle information interface is still allowed to be used by the decision component. The decision component still requires information from vehicle control and there is no reason to have vehicle information go through the safety component as it will not be altered in any way.

The safety component will provide an interface which can be used to check which decisions are considered safe at the moment. Although this is not necessary for the separation, the information provided by the interface could be used to develop a decision component which does not propose unsafe actions. If a decision component does not consider safety aspects and have no access to allowed actions, the proposed actions would frequently be altered by the safety component.

The safety component has to rely on the perception and vehicle control components. If the output from perception or vehicle control is wrong, or if actions are not executed properly, the safety component will not be able to ensure safety. This means that only the decision component is relieved of responsibility from safety aspects, perception and vehicle control will keep their safety requirements.

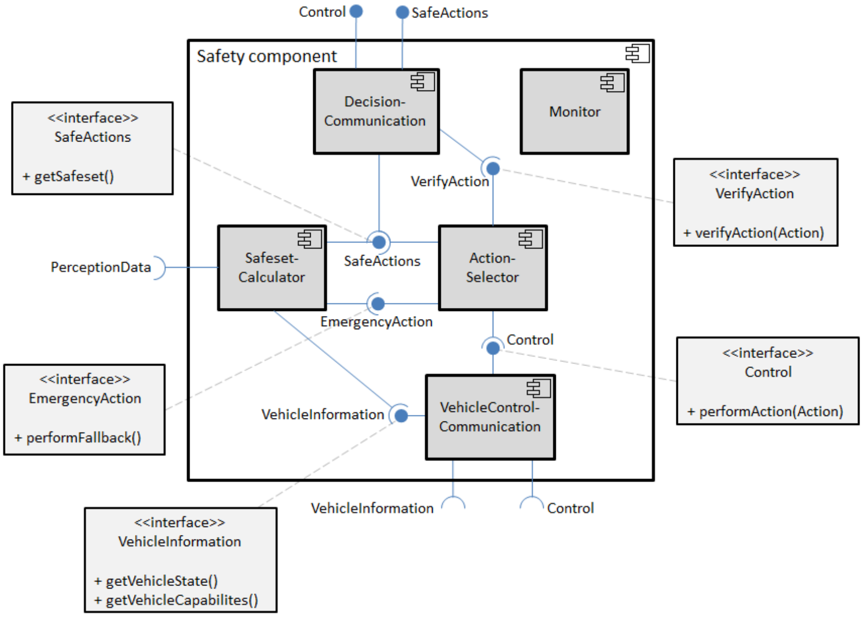

Control
SafeActions
Safety component
Decision-Communication
Monitor
<<interface>>
SafeActions
+ getSafeset()
VerifyAction
<<interface>>
VerifyAction
+ verifyAction(Action)
PerceptionData
Safeset-Calculator
SafeActions
Action-Selector
EmergencyAction
Control
<<interface>>
Control
+ performAction(Action)
<<interface>>
EmergencyAction
+ performFallback()
VehicleInformation
VehicleControl-Communication
VehicleInformation
Control
<<interface>>
VehicleInformation
+ getVehicleState()
+ getVehicleCapabilites()

Figure 5.3: Top level architecture of the safety component. The connections to *Monitor* is left out as it is connected to all other components.

5.3 Safety Component

The top level architecture of the safety component consists of five different subcomponents, each with its own responsibility. The architecture can be seen in Figure 5.3. The different responsibilities is shortly described in this section before describing the design in more detail in the following sections.

SafesetCalculator is responsible for calculating whether different actions can be considered safe or not. An action in this context is a combination of longitudinal acceleration and steering angle. This is done using environment data available in the perception component as well as the vehicle's capabilities available in the vehicle control component.

SafesetCalculator should run before or in parallel with the decision component. This means that a safe action already should be known when it is time to test the action given by the decision component. As a result, the system does not have to spend additional time finding an action if the decision component's action is deemed unsafe. Having quick verification supports requirement GR3 of having real-time execution. Furthermore, if the safeset already contains the action proposed by the decision component the verification time would be low. However, this would require *SafesetCalculator* to preemptively check a significant number of potential actions. This would differ to the approaches described in section *4.1 Layer* by Underwood et al. [12], Goebl et al. [30] and Gindele et al. where a trajectory is verified upon request as opposed to in advance.

If the verification occurs upon request the average time to verify an action would be longer but significantly less resources would have been used as *SafesetCalculator*

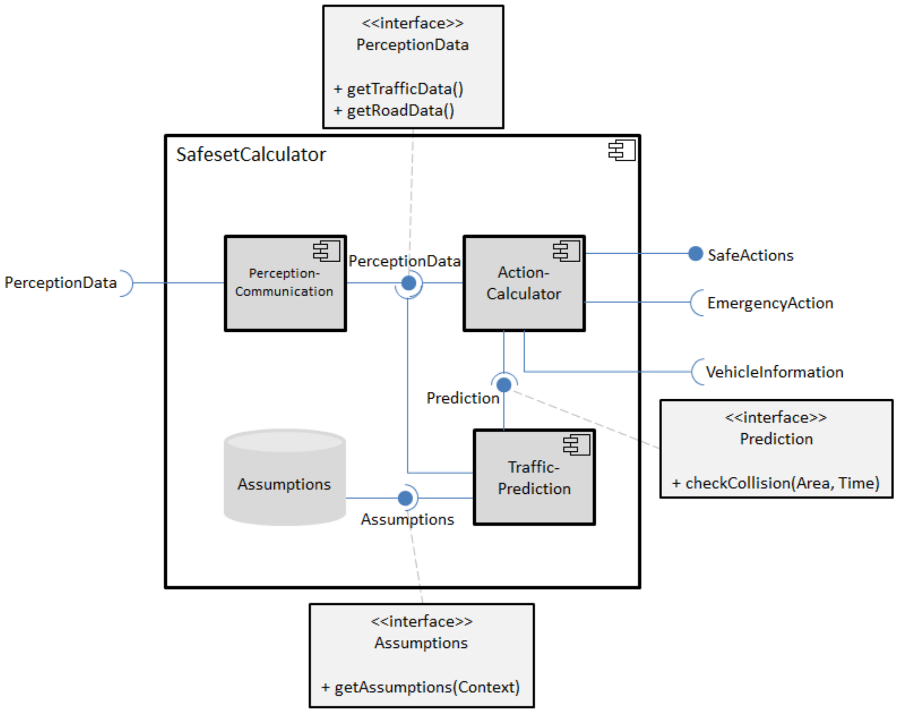

<<interface>>
PerceptionData
+ getTrafficData()
+ getRoadData()
SafesetCalculator
PerceptionData
Perception-
Communication
PerceptionData
Action-
Calculator
SafeActions
EmergencyAction
VehicleInformation
Prediction
<<interface>>
Prediction
+ checkCollision(Area, Time)
Assumptions
Traffic-
Prediction
Assumptions
<<interface>>
Assumptions
+ getAssumptions(Context)

Figure 5.4: Diagram of the subcomponents inside *SafesetCalculator*.

does not have to test every action. Although, finding a safe action could in worst case mean that every possible action have to be tested. In that case the verification of the proposed action would be as fast as in the first approach, but the total resource usage would also be as high.

Running *SafesetCalculator* in advance also means that the decision component could make use of the information regarding which actions are currently considered safe, this is done through the interface *SafeActions*. *DecisionCommunication* provides this interface for the decision component and is also responsible for receiving proposed actions from the decision component. When the decision component sends a proposed action for the vehicle to perform, the communication component forwards the action to *ActionSelector*. *ActionSelector* verifies that the action will not lead to an accident by consulting *SafesetCalculator*. After verifying the action *ActionSelector* sends it to *VehicleControlCommunication*, which is responsible for sending the message to the vehicle control component.

Monitor tries to detect when a component fault occurs. By watching the different components the monitor can detect when one fails or malfunctions which allows the monitor to respond accordingly.

5.3.1 Safeset Calculator

Figure 5.4 shows the subcomponents of *SafesetCalculator*, which responsibility is to calculate actions that currently are safe to perform. A set of safe actions will henceforth be referred to as safeset. Depending on the implementation the component could stop trying to find additional actions any time after it has found at least one safe action. Having at least one safe action is necessary for the vehicle to have a safe action to perform in case the decision component sends an unsafe action. In the event that no safe action can be found *ActionCalculator* can use a solution based on the instant action pattern, see section *4.3 Instant Action*, and directly call for an emergency fallback through *EmergencyAction*.

Calculating which actions are safe is done by first acquiring information about the environment from the perception component. For each perceived traffic participant a reachable area is calculated, similar to the approach presented by Goebl et al. [30]. The area is calculated using the participant's current state and assumed limits on how it can accelerate within a certain time frame.

After generating the reachable area for each traffic participant, a number of assumptions are applied. If the reachable area does not take any assumptions into consideration it must give all positions a participant can reach, even if they are unlikely. This will quickly limit the number of safe actions as the reachable area for a fast moving traffic participant, such as a passenger car, would fill most of the road surface after just a few seconds. An example of an assumption to make could be that a vehicle will not accelerate in such a way that it will drive into the rear end of the ego vehicle. By using assumptions on how traffic participants can move the reachable area can be limited and thus open for more actions to be considered safe. However, defining acceptable assumptions is not in scope of this study.

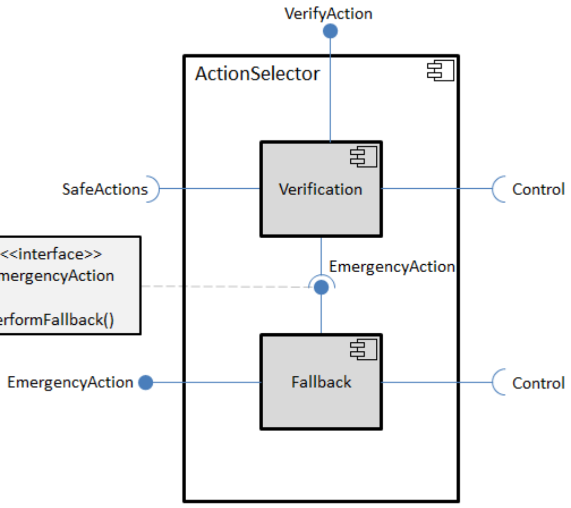
To support requirement GR7, of being able to handle different traffic situations, sets of assumptions could be specified for different driving environments. An example of such an assumption could be that a vehicle will not be going the wrong way down a highway. Requirement GR5, of supporting different vehicle behavior, will also be supported by having assumptions as they can enable the vehicle to perform more, or less, actions.

The predicted reachable area for all traffic participants are used to decide which actions to put in the safeset. Given the capabilities, from the vehicle control component, potential paths can be tested by *ActionCalculator*. If a path does not intersect any of the reachable areas then the action that goes along the path can be added to the safeset. Having an approach which can calculate safe actions given different capabilities supports requirement GR6 of being vehicle independent.

5.3.2 Action Selector

ActionSelector is decomposed into two different sub components, see Figure 5.5. The normal flow is that *Verification* receives an action from the decision component through *DecisionCommunication*. *Verification* checks that the action is safe using the safeset provided by *SafesetCalculator*, if the safeset contains the action then it is safe by definition and can be forwarded to *VehicleControlCommunication*.

Figure 5.5: Diagram of the subcomponents inside *ActionSelector*.



There can be two reasons for an unsuccessful verification, either the action from the decision component does not exist in the safeset or no action is being received by *Verification*. In both cases *Fallback* is being responsible for choosing an action quickly. If the safeset is empty *Fallback* still needs to decide on an action, for example maximum brake, as it is the last resort for handling a situation before passing the problem to vehicle control. However, *Fallback* should only have to be called if the decision component fails or something occurs in the traffic which was out of scope during the design of the system. An example of something out of scope could be a tree falling onto the road right in front of the vehicle.

In the case where the action from the decision component is not considered safe there is an alternative approach to calling *Fallback*. Instead, the closest safe action could be chosen. For example, it might be necessary to break if the desired action is to continue forward. If this approach is chosen it is necessary to define how to select the closest action as well as how far off the decision component is allowed to be before the vehicle should stop.

ActionSelector is always responsible for each action which is being sent to vehicle control communication. Either the action is verified by *Verification* or the action is being decided by *Fallback*. This allows the safety component to ensure the safety of the vehicle independently of the behavior of the decision component, according to requirement GR2.

5.3.3 Redundancy

To support requirement GR4, of being fail-operational, redundancy of the components is used, see Figure 5.6. As described in section *1.4 Scope*, even though the study focuses on the software aspect of the problem it is still necessary to create a design which allows for hardware redundancy. Different approaches to using redundancy are described in section *4.4 Faults and Redundancy*.

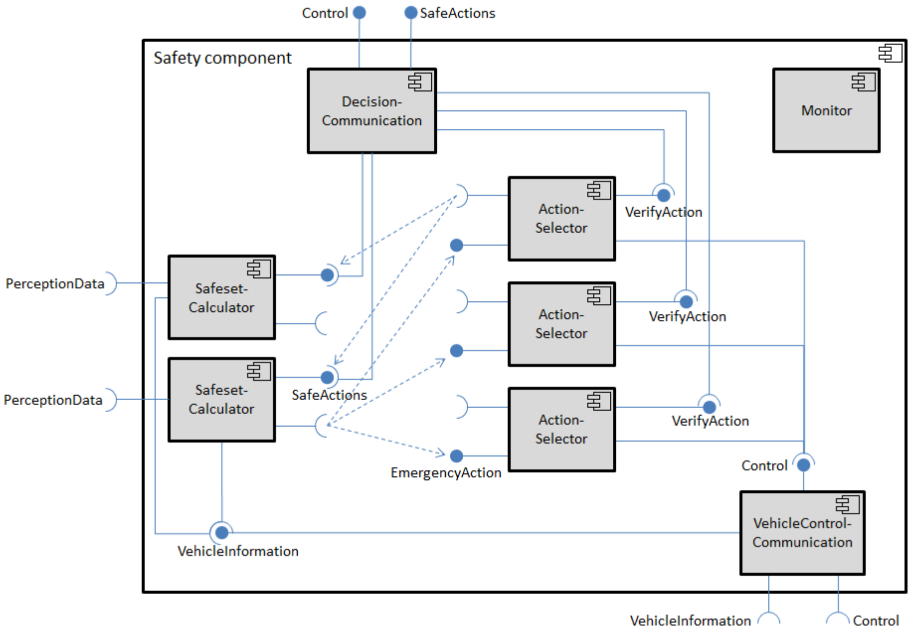

Control
SafeActions
Safety component
Decision-Communication
Monitor
Action-Selector
VerifyAction
Action-Selector
VerifyAction
Action-Selector
VerifyAction
PerceptionData
Safeset-Calculator
PerceptionData
Safeset-Calculator
SafeActions
EmergencyAction
Control
VehicleControl-Communication
VehicleInformation
VehicleInformation
Control

Figure 5.6: Diagram showing how redundancy could be implemented. The connections to *Monitor* is left out as it is connected to all other components. Also note that not all connections between the instances of *SafesetCalculator* and *ActionSelector* are shown.

The relation between the instances of *SafesetCalculator* can be considered as an AND-gate and thus it only needs to run in double redundancy. If *ActionSelector* verifies an action against both instances of *SafesetCalculator* then both must consider an action to be safe before it can be executed by the vehicle. In the case that one of the instances fail in such a way it consider safe actions as unsafe then *ActionSelector* will not be able to verify the decision component's action and thus use its fallback solution. In the other case when one of the instances fail such that it consider unsafe actions as safe the safeset will still be limited to the actions given by the functioning instance. It is important to note that double redundancy of *SafesetCalculator* can only be used if the fallback solution is considered safe enough given the risk that an *SafesetCalculator* fails. The alternative is to use triple redundancy with voting.

The output of *ActionSelector* is an action for the vehicle control component to execute. The component will run in triple redundancy with voting. As a result of this the output of *ActionSelector* must be deterministic in some sense as the voting mechanism needs to get at least two identical actions for it to be able decide the majority vote. As the voting mechanism only needs two identical actions one of the instances of *ActionSelector* can fail without impacting the operation of the vehicle. The constraint of enforcing deterministic output for *ActionSelector* is not a problem for *Verification* as it only checks whether a certain action is safe. However, it puts some limitations on how *Fallback* can be implemented.

Due to the fact that one instance of *SafesetCalculator* and one instance of *ActionSelector* can fail without making the whole safety component fail means that the safety component as a whole can be considered fail-operational, thus supporting requirement GR4. However, if additional instances fail the safety component can no longer operate safely. This means that when a failure is detected some mechanism, in this case the monitor, should ensure the system either recovers or safely stops.

As *SafesetCalculator* and *ActionSelector* run in redundancy the communication to components outside the safety component must be handled differently. *DecisionCommunication* compiles the safesets from both instances of *SafesetCalculator* before providing it to the decision component. It also sends an action proposed by the decision component to all instances of *ActionSelector* so that the decision component does not need to concern itself regarding the redundancy. The vehicle control component is assumed to only take one action as input and thus a component is needed to combine the output from the instances of *ActionSelector*. This component, *VehicleControlCommunication*, should also perform the voting of which action to send.

5.3.4 Monitor

In order to meet requirement GR4, to be fail-operational, the system needs to be able to detect faults and when necessary take precautions. *Monitor* requests status updates from different components and checks for faulty behavior, similarly to how it is described in section *4.2 Monitor and Watchdog*. One way to discover faults could be to check if *VehicleControlCommunication* continuously receives results from one *ActionSelector* which differ from the other's results. This would be an indication that one *ActionSelector* is failing.

The monitor also has a responsibility to take action if there is a critical fault. This could include restarting a faulty process or initiating an emergency stop. If a critical fault occurs it could for example be necessary to stop the vehicle and do a full system restart.

6

Industrial Case

This chapter presents the result of applying the general architecture from chapter *5 General Architecture* in an industrial case at Volvo Group Trucks Technology. The requirements for the safety component will be presented as well as the resulting architecture. A prototype of the safety component has also been implemented and will be presented together with simulation results.

This results presented in this chapter was used to improve the general architecture in chapter *5 General Architecture.* Thus, it was also used to improve the answer to RQ2, to propose a general architecture which separates safety constraints and functionality in the context of autonomous road vehicles.

6.1 Requirements

The requirements for the safety component have been divided into four categories: context, safety, performance and architecture. The requirements have been elicited and specified according to what is described in section *3.3.1 Requirement Elicitation.* This section only summarizes the requirements while the actual requirements are specified in *Appendix B Industrial Case Requirements.*

Context requirements relates to the context in which the system should be applicable. The requirements handle the fact that the system should not be dependent on a specific vehicle configuration or traffic situation. It is also necessary for the vehicle to be able to behave differently depending on the current context and to be able to switch between different contexts while running.

The safety component needs to be able to handle all kinds of safety concerns which could arise as a result of driving decisions. Even if the decision component is offline or starts proposing dangerous actions the safety component should be able to ensure safe behavior of the vehicle. The safety component must ensure that an action is safe and thus take full responsibility for the action before allowing it to be carried out.

Performance requirements are important in the sense that the safety component needs to be able to run with real-time execution and with low delay. However, performance is dependent on both the implementation and available hardware resources. While the hardware resources should be minimized in order to save costs there is no hard limit, which leads to fuzzy requirements.

The safety component also has some requirements focusing on the architecture as a whole. This includes requirements regarding how the safety component is allowed to, and in some cases should, impact surrounding systems. As the safety component should be able to ensure safety on its own it should be able to take responsibility of all safety constraints related to decision making. As a result the decision component should not have any formal safety constraints.

Requirements can have an effect upon each other. Satisfying many negatively impact another. These requirements need to be identified and analyzed. In this study, primarily one case demanding special attention was identified.

The requirements to have high performance and to allow safe actions can be problematic to implement at the same time without making some sort of trade-off. For example, in an extreme case where performance is highly prioritized *SafesetCalculator* must quickly decide whether an action is safe or not. However, since it is crucial no actions are falsely considered safe *SafesetCalculator* cannot simply guess and must find evidence that the action is safe to perform. As finding such evidence in a performance efficient way could be difficult in many scenarios, the resulting safeset would often contain few or no safe actions. In contrast, if only precision is prioritized, the approach could be to test every potential path or recognize every potential situation the vehicle might be in. As a result, if a safe action complying with safety constraint exists in reality, it would be unlikely that *SafesetCalculator* would fail to find it.

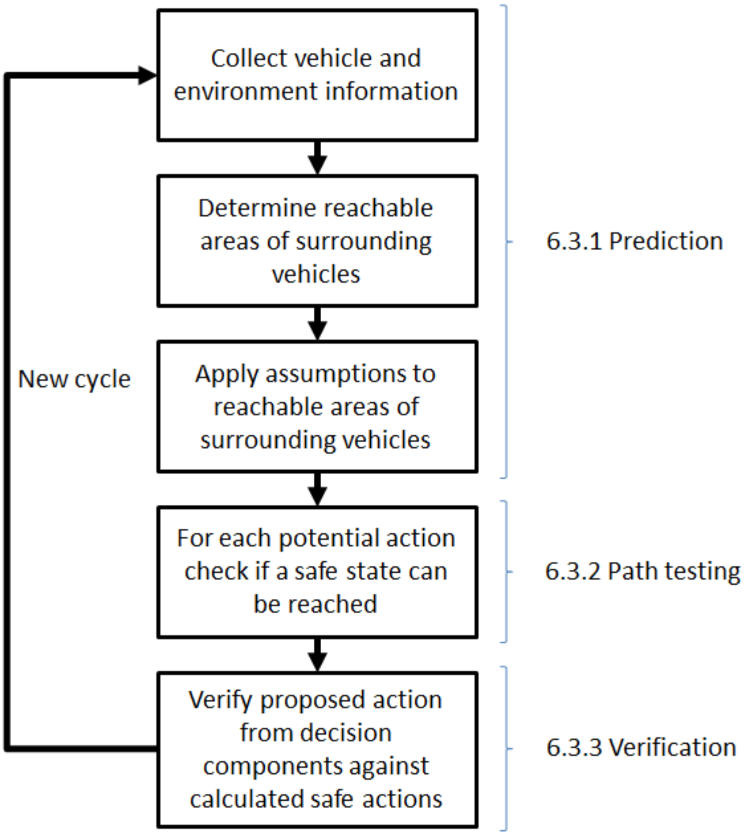
Allowing as many safe actions as possible could be translated to the precision which is used when deciding whether an action is safe or not. This is because less precision means that larger error margins are required as safety needs to be ensured whether the algorithm is precise or not. These requirements correspond to P1-P6, performance, and S2, allow safe actions, in *Appendix B Industrial Case Requirements*.

6.2 Safety Component

This section describes how the general architecture for the safety component was applied in order to meet the requirements for the industrial case. It also presents an overview of the used solution. The solution does not aspire to be the only viable solution, but instead aims to show the feasibility of using the architecture.

In the proposed solution, *SafesetCalculator* must always find at least one safe action which can be used if the proposed action, from the decision component, is unsafe. Calculating a safe action is not something which necessarily can be done in a short

Figure 6.1: The process, within the safety component, of defining safe actions and verifying a proposed action from the decision component.

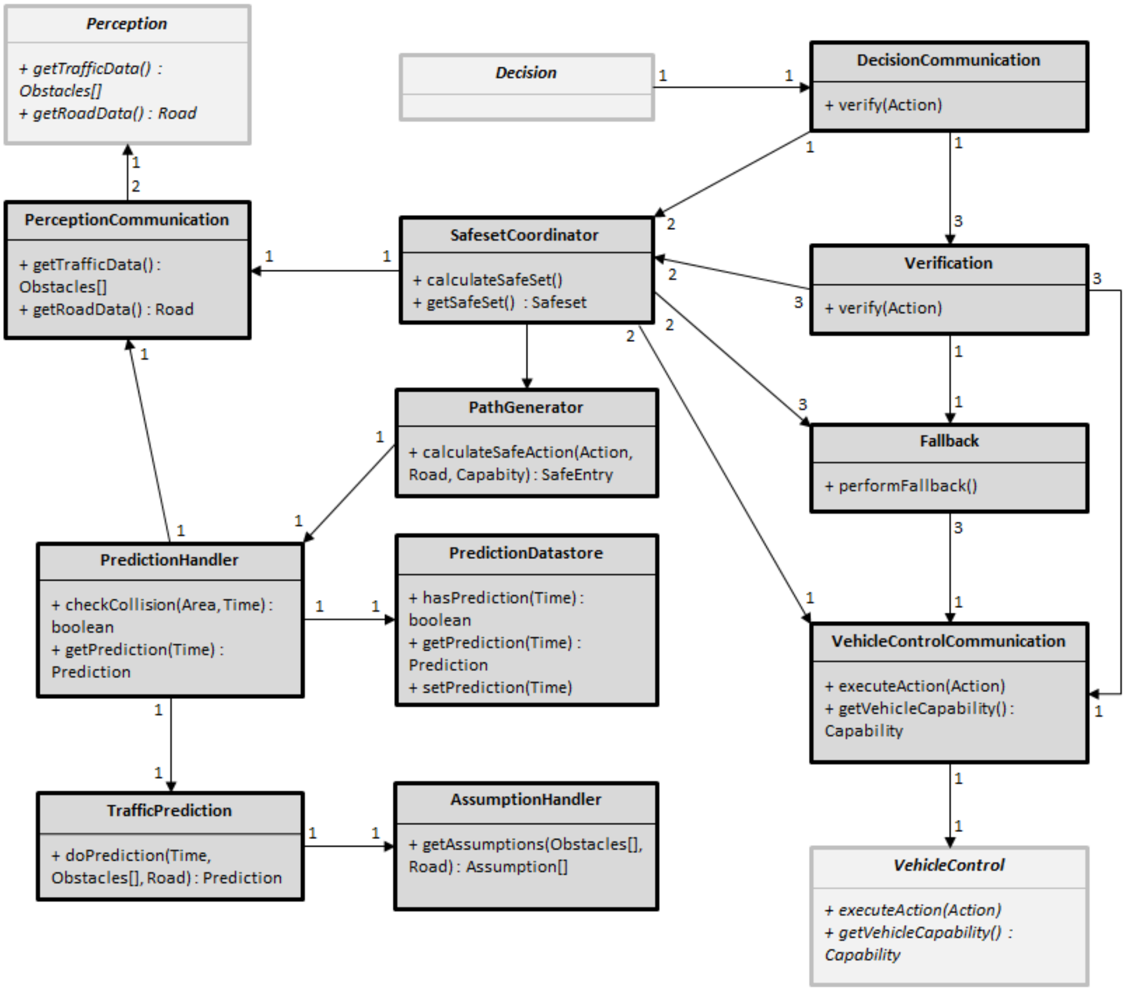

Collect vehicle and environment information
Determine reachable areas of surrounding vehicles
Apply assumptions to reachable areas of surrounding vehicles
For each potential action check if a safe state can be reached
Verify proposed action from decision components against calculated safe actions
New cycle
6.3.1 Prediction
6.3.2 Path testing
6.3.3 Verification

time frame. In order to minimize the delay from receiving an action to sending an action to vehicle control, safe actions are calculated before, or in parallel with, the decision component.

SafesetCalculator provides information regarding which actions are considered safe to other components. The approach of calculating all safe actions, as opposed to only verifying a specific action, does not necessarily increase the amount of hardware resources needed. Since calculating a single safe action in a worst case scenario could require as much resources as calculating a complete safeset. The result is an approach where *SafesetCalculator* calculates all safe actions, share the information and verifies suggested actions against an already calculated safeset.

The proposed process of the safety component can be seen in Figure 6.1. The process can be divided into three steps: prediction, path testing and verification. These steps are presented in the following sections. Furthermore, an abstract class diagram, see Figure 6.2, with the most central classes in the safety component is used when describing the different steps. Finally, there is a section discussing which requirements the proposed solution supports.

Figure 6.2: Abstract class diagram with the most central classes in the safety component. Three classes are written in italics and represent the connection to surrounding components.


Perception
+ getTrafficData() : Obstacles[]
+ getRoadData() : Road
Decision
DecisionCommunication
+ verify(Action)
PerceptionCommunication
+ getTrafficData() : Obstacles[]
+ getRoadData() : Road
SafesetCoordinator
+ calculateSafeSet()
+ getSafeSet() : Safeset
Verification
+ verify(Action)
PathGenerator
+ calculateSafeAction(Action, Road, Capabity) : SafeEntry
Fallback
+ performFallback()
PredictionHandler
+ checkCollision(Area, Time) : boolean
+ getPrediction(Time) : Prediction
PredictionDatastore
+ hasPrediction(Time) : boolean
+ getPrediction(Time) : Prediction
+ setPrediction(Time)
VehicleControlCommunication
+ executeAction(Action)
+ getVehicleCapability() : Capability
TrafficPrediction
+ doPrediction(Time, Obstacles[], Road) : Prediction
AssumptionHandler
+ getAssumptions(Obstacles[], Road) : Assumption[]
VehicleControl
+ executeAction(Action)
+ getVehicleCapability() : Capability

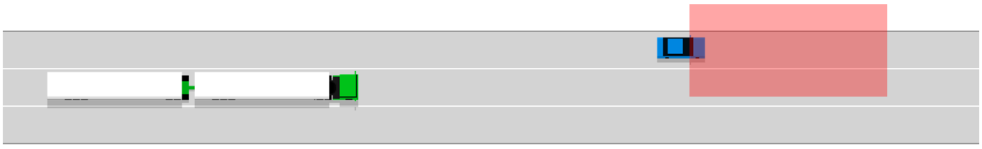


Figure 6.3: The reachable area of a traffic participant has been marked in red. In this case, the reachable area has been simplified to a rectangle and is only used to illustrate the principle.

6.2.1 Prediction

The class *TrafficPrediction*, see Figure 6.2, is responsible for calculating where another traffic participant could be a specified amount of time in the future. A prediction consists of a set of reachable positions, or in other words reachable area, for a traffic participant detected by the ego vehicle. A simple representation of the reachable area can be seen in Figure 6.3 However, the system needs to consider that there potentially could be a traffic participant just outside the detection area. This means that if the vehicle is able to detect other vehicles within x meters, it should be considered that there could be a vehicle $x + 1$ meters away.

The predictions are only calculated for as long time as necessary for the ego vehicle to reach a safe state. The class generating and checking paths, *PathGenerator*, requests predictions from the class *PredictionHandler* of different time steps. As the paths are only tested until the vehicle reaches a safe state no predictions needs to be done for later time steps. The predictions are cached to avoid calculating the same prediction twice when testing different paths. When *PathGenerator* requests a prediction either the cached version is returned or, if no cached prediction exists, a new one is calculated.

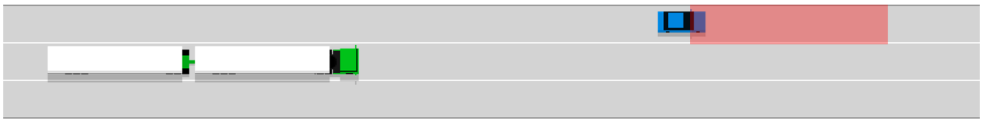


Figure 6.4: The reachable area of the traffic participant from Figure 6.3 has been limited to the lane of the traffic participant.

As described in section *5.3.1 Safeset Calculator* assumptions are used to limit the size of the reachable area for a traffic participant, see Figure 6.4. Assumptions are calculated independently of both collision checking and reachable areas of traffic participants, meaning assumptions can easily be changed both online and offline. This is useful as the assumptions can be changed depending on context, such as type of traffic participant, type of road and geographic location with different traffic rules. However, this does not mean that the assumptions can be switched without proper safety verification. They are part of the safety component and have a large impact on how the vehicle behaves. In order to be able to switch between different assumptions each assumption needs to be properly verified for in which contexts it can be used.

Since assumptions limit the reachable areas of traffic participants more assumptions generally mean that more safe actions can be found. Thus using assumptions result in that there are situations which are theoretically possible, but are not taken into consideration when planning a path. Inevitably assumptions will decrease the safety of the vehicle, but without assumptions it will be more difficult for the vehicle to reach its destination without delay. For example, without assumptions the vehicle would have to take into account that there could be a vehicle going against the direction of the road right outside the detection area. If the detection area is small the vehicle would not be able to drive with a significant speed. This study is cautious when making this trade-off and only uses a few assumptions during the industrial case. The reason for this is that defining safe assumptions of the behavior of other traffic participants is outside the study's scope.

There are some base assumptions limiting the maximum velocity and acceleration of vehicles, however, these assumptions are more connected with physical limitations than driver behavior. It is also assumed that no vehicle will go against the direction of the road, drive into the ego vehicle from behind and switch lane into the side of the ego vehicle given that the ego vehicle is staying in its lane. After assumptions have been applied to all traffic participants' reachable areas, they are made available for when different paths are tested.

6.2.2 Path Testing

The reachable areas of other traffic participants are used when calculating which actions are considered safe. An action is defined as safe if and only if it is possible for the vehicle to reach a safe state without an accident occurring after performing the action. Similarly, a safe state is defined as a state which can lead to another safe state without an accident occurring. In this study the only state which is considered safe by default is when the vehicle is standing still. As a result, in this study a vehicle is in a safe state if it can safely stop.

Initially the class *SafesetCoordinator* uses the current vehicle state and potential actions to calculate possible states the vehicle can be in at the beginning of next cycle. The possible states are passed to the class *PathGenerator* which decides whether or not it is possible to reach a safe state from each potential vehicle state. If it is possible to reach a safe state the action which lead to the state is considered safe.

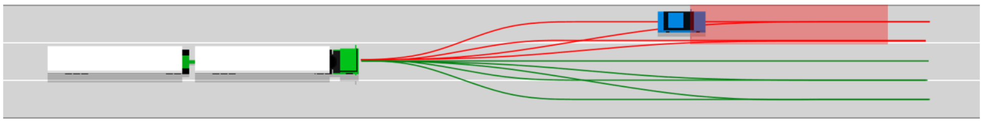


Figure 6.5: Here a number of tested paths can be seen. Paths which are not considered safe are marked in red and paths which are considered safe are marked in green.

The class *PathGenerator* generates and tests a number of paths for each state, see Figure 6.5. Ideally, different paths should be generated depending on the context, such as type of road and road curvature. Paths which are adjusted to the context could increase the chance to find safe paths and thus allowing the decision component to choose between more actions. While testing more paths will increase the chance to find more safe actions it is also important to consider that more paths will require more processing power as well.

Each path is evaluated in small time steps. For each time step the distance traveled is calculated using the velocity and maximum available deceleration. The distance together with different margins is used to calculate an area which is checked for collision in the time step currently evaluated. This means that the class *PathGenerator* asks the class *PredictionHandler* if there is a risk for collision in the area. Four margins are considered: the size of the vehicle, the offset between the vehicle and the planned path, the distance traveled between brake request and brake application and a general error margin. The general error is a margin which is added partly to avoid stopping too close to another vehicle and partly to reduce the risk from unpredicted faults. Some of the margins could have been removed or significantly smaller if accurate vehicle models were used to calculate the exact vehicle position in each time step. However, using precise models in these cases would also significantly increase the calculation time, which in turn could force *PathGenerator* to test less paths to be able to run in real-time.

6.2.3 Verification

The set of actions which are deemed safe during path testing create a safeset. The class *Verification* compares proposed action from the decision component to the safeset. If the action is safe, it is passed to the class *VehicleControlCommunication* and if the action is not safe, *Verification* will choose the closest safe action. The closest action is determined by looking at the safe action with the closest steering angle. However, if the closest action is chosen the safety component will also request that the vehicle brakes. This is because it is considered better to slow down as the steering of the vehicle is no longer deliberate from a non-safety point of view, in other words there is no guarantee that the vehicle steers towards its destination.

If the decision component proposes actions which are not allowed several times in a row the safety component will stop listening on proposed decisions and completely stop the vehicle. It is considered better to stop the vehicle than it is to attempt to follow a faulty decision component. The class *Fallback* will only be called when there is no safe action and, in the current implementation, always requests maximum brake from the vehicle control component.

6.2.4 Supported Requirements

All requirements referenced in this section can be found in *Appendix B Industrial Case Requirements*. Requirements S1-S6 together means that the safety component should be able to choose, and allow, safe actions at all times and to take responsibility that only safe actions are sent to the vehicle control component. The responsibility is fulfilled through the architecture as the safety component is the only one allowed to use the vehicle control component's control interface. Choosing, and allowing, safe actions is ensured through the process in Figure 6.1. This process also enables the safety component to provide information about which actions currently are considered safe, see requirement S7.

By taking full responsibility for safety the requirements A3 and A4 are also fulfilled, the decision component has no safety responsibility and thus no ASIL requirements. The safety component uses the same interfaces for the same reason as a decision component with safety responsibility would do, thus the ASIL requirements on external components does not increase, see requirement A1. The internal ASIL requirements, see requirement A2, are minimized through redundancy as described in section *5.3.3 Redundancy*.

Requirement C1 specifies that the vehicle should be able to handle different traffic environments. This can partly be supported by letting *SafesetCalculator* test different paths for different environments. For example, an intersection and a highway require different paths. The other part of C1 is solved by allowing different sets of assumptions to be used in different contexts, something which also fulfills requirements C3 and C4 which specify that the system should be able handle and change the assumed behavior of other traffic participants. Requirement C2 is handled by using vehicle capability limits, the *VehicleInformation* interface, from the vehicle control component when determining which actions are possible to make at a certain point in time. For example, if the vehicle can brake with a certain force the safety component does not test if it is safe to brake with a higher force.

Requirements P1 and P3-P6 specify that the system should be able to run in realtime using as little hardware resources as possible. It is hard to determine whether these requirements are fulfilled as there is no hard limit. However, the required hardware resources can be varied depending on how much precision is wanted, see section *6.1 Requirements*. Requirement P2 is related to the performance requirements and specifies that there should be a low delay between receiving a decision and making a decision. This requirement is addressed through calculating safe actions in advance, see section *5.3 Safety Component* for more information.

6.3 Prototype

While the architecture aims to be applicable in different situations the implementation has a more narrow scope. The implementation is a prototype acting as a proof of concept. The prototype is limited to a number of highway scenarios and one type of vehicle. The vehicle is a 32 meter long vehicle combination consisting of a tractor, semi-trailer, dolly and another semi-trailer.

6.3.1 Implementation Scope

All requirements discussed in section *6.2.4 Supported Requirements* are implemented in the prototype with a few exceptions. The redundancy of *SafesetCalculator* and *ActionSelector* was not implemented in such a way that it runs on different hardware. Instead, the redundancy was implemented to run as a single process meaning timing issues and other synchronization problems were not considered. However, with the current implementation it is possible to test how the system behaves when instances of *SafesetCalculator* and *ActionSelector* are simulated to fail.

The monitor is supposed to detect failing processes, but was not implemented. The main reason to have a monitor is to be able to detect when something goes wrong and then perform an appropriate action given that information. Due to the limited scope, and the fact that the redundancy could be tested without a monitor, it was left out of the prototype. However, in a more mature system the monitor would have been added as it could give valuable information when testing the system.

Due to the scope of only supporting a number of highway cases in the prototype *PathGenerator* only generates paths suited for highway driving. This means that all paths end in the same direction as the road. The paths are generated using the ISO 14791 [34] definition of a lane change with various longitudinal and lateral distance. In other traffic contexts different algorithms for generating paths could be used to ensure the safeset does not become too restrictive. Furthermore, even in highway cases the paths could be improved to generate more diverse paths.

6.3.2 Scenarios

The prototype was tested in nine simulated scenarios specified in *Appendix A Highway Scenarios*. A summary of the result can be seen in Table 6.1. All scenarios are in the context of highway driving, meaning there is no oncoming traffic. It is also assumed that no vehicle will drive into the rear end of the ego vehicle if they are in the same lane. Some scenarios are simulated both with and without an assumption where vehicles never change lane. While this assumption might not be safe to include in a real scenario it is used here to showcase how other safety constraints limit allowed actions.

Scenarios on a road with only one lane gives simple tests as traffic in other lanes do not have to be considered. The most basic test is having one lane without any other traffic than the ego vehicle itself, see scenario 1. Simulation results show that the prototype does not alter the decision components actions as long as the vehicle stays

Table 6.1: Summary over results of scenarios.

Scenario	**Without assumptions**	**With assumptions that other vehicles stay in their lane**
1	Safe at current speed	Safe at current speed
2	Safe at current speed	Safe at current speed
3	Safe at other vehicle's speed	Safe at other vehicle's speed, or lane change at current speed
4	Safe at current speed	Safe at current speed
5	(Depends on position of other vehicles)	(Depends on position of other vehicles)
6	Unsafe	Safe at current speed
7	Unsafe	Safe at current speed
8	Unsafe	Safe at current speed
9	Emergency brake	Emergency brake

on the road. When the ego vehicle tries to leave the road the prototype interferes to keep the vehicle in a safe state. If another vehicle is placed in front of the ego vehicle, see scenario 2, the prototype ensures that a safe distance is kept even when the ego vehicle tries to decrease the distance. If an empty parallel lane is added, see scenario 3, the prototype also allows the ego vehicle to perform a lane change to overtake the vehicle. However, the prototype only allows the ego vehicle to overtake if it can assume that the other vehicle stays in its lane.

If there are no surrounding vehicles and the ego vehicle tries to make a lane change, see scenario 4, the prototype does not alter the actions as long as they do not result in leaving the road. In case there are vehicles in the lane targeted to change lane into, see scenario 5, the prototype will in not allow the ego vehicle to go too close to the target lane unless there is large abundance of available space in the lane. Even if there is enough available space at the moment the worst case prediction is that the vehicle in front brakes and the vehicle behind accelerates, which results in the prototype requiring large margins in order to consider a lane change safe. To be able to make lane changes in the same way normal drivers do, it would be necessary to have assumptions on how vehicles in other lanes behave.

Without assumptions regarding how another traffic participant is allowed to change lane, the ego vehicle is not able to overtake another traffic participant, see scenario 6. Since the vehicle being overtaken could change lane right in front of or into the side of the ego vehicle it will, in most cases, be considered unsafe to overtake another vehicle. Exceptions can occur if the velocity is low enough, for example if overtaking a queue as in scenario 7. However, in simulations where other vehicles are assumed to remain in their lane the ego vehicle can overtake other vehicles without the prototype intervening.

There are similar problems when another vehicle overtakes the ego vehicle, see scenario 8. Since other vehicles could potentially change into the ego vehicle's lane it is not considered safe to have another vehicle next to the ego vehicle. However, if the vehicle is assumed to not change lane, the prototype will not intervene. A problem which could occur using such an assumption is that the other vehicle could change lane in front of the ego vehicle closer than what is considered safe, see scenario 9. This would trigger *Fallback* which is something that should not happen during normal circumstances. Since it is reasonable to expect that other vehicles will sometime change into the ego vehicle's lane close to the ego vehicle, it is not always safe to assume that other vehicles will stay in their lane.

6.4 Evaluation of Results

The simulation results of the prototype show that it can be used to avoid accidents in highway scenarios. However, without assumptions the ego vehicle will in many scenarios not be able to drive at any significant speed. As a result, it is necessary to investigate what assumptions are safe if the system is ever to be used in an environment with other traffic participants. It should be noted that defining assumptions are out of scope for this study as it would be necessary to determine the behavior of other traffic participants whether the decision component is separated or not. This is further discussed in section *7.2 Assumptions*.

In the architecture the fallback action is specified to be handled by a component called *Fallback*. The implementation of *Fallback* used in the prototype is simple and the functionality could have been handled by *Verification* instead. However, the complexity of *Fallback* could significantly grow in the future. An example would be if *Fallback* was supposed to select the action leading to the least severe accident. *Fallback* would then have to be able to assess the outcome of accidents involving different types of traffic participants in different road contexts.

The architecture and highway scenarios were presented to four industry experts, as described in section *3.3.3 Evaluation*. The expert in software architecture focused on the architecture more than the prototype while the experts in active safety and vehicle dynamics primarily focused on the overall solution together with the prototype and scenarios.

All the experts considered the solution with division of responsibility between functionality and safety constraints as a promising approach. The software architect expert pointed out that the architecture would solve problem with ASIL requirements locally on the decision component. However, he also pointed out that it might not be needed if the ASIL problems on the overall architecture could be solved. Having no ASIL requirements on the decision component might not be the most probable solution as opposed to aiming for ASIL B, or lower, using redundant decision components.

The active safety expert emphasized that the division of responsibility between functionality and safety constraints would be beneficial during the system's life cycle. He pointed out that the cautious approach of never allowing an action unless an assumption has been specified to allow it, makes it possible to see exactly why a certain action was allowed. Another appreciated feature was that the decision component could request all currently safe actions as this potentially could make it easier to implement well-functioning independent decision components. Furthermore, he mentioned that there is a potential risk that the safety component will go directly from being too difficult and expensive to being unnecessary due to technical advances. However, he considers the separation of functionality and safety constraints to be a highly relevant topic in vehicle automation and that the proposed approach could be one way of doing it.

The vehicle dynamics experts considered the problems which prevent the solution from being deployed on a real vehicle. Most problems they brought up are problems which must be solved regardless of whether the proposed safety component is added or not. The rest of the problems they considered were related to the fact that the prototype did not contain or simplified features which should be included in a complete implementation. Accordingly, they would have preferred a prototype which used a more advanced model of the vehicle and thus reach a higher precision in calculations. Another mentioned issue was that the proposed solution, depending on implementation, can result in some calculations having to be done twice, once by the decision component and once by the safety component. However, they agreed with the benefit of having no ASIL requirements on the decision component.

The experts in vehicle dynamics and active safety agreed that the proposed safety component could function as a tool to see what needs to be improved or changed in order for the ego vehicle to behave satisfactory in different situations. Primarily, this could be used to see where the current assumptions do not allow the vehicle to behave as desired. If it is possible to identify where assumptions are needed it could be used to specify which driving environments the vehicle can operate in. It could also be used to determine what is required in order for the vehicle to drive in other environments.

7

Discussion

This chapter discusses the result and its limitations. Initially there is a discussion regarding the outcome of the study and the proposed approach. It is followed by potential applications of the proposed approach and a discussion regarding important trade-offs. The last sections bring up issues regarding single point of failure and the limitations of the study.

7.1 Outcome

Underwood et al. [12], Goebl et al. [30] and Gindele et al. [31] all suggest approaches where a safety layer should verify all actions before they are carried out. Underwood et al. and Gindele et al. also specify that the safety layer is responsible for generating a new action if the verification process deems an action unsafe. However, none of these specify how the separation could be done. Only Underwood et al. discuss in terms of giving the safety component complete responsibility over safety and possibly removing all safety requirements from decision components. This study suggests an approach with a similar intent as what is described by Underwood et al.

A key aspect of the approach suggested by this study is the division of responsibilities described in section *5.2 Responsibilities.* The safety component has no responsibility to ensure that the vehicle reaches its destination and no responsibility that the trip is comfortable for passengers. The decision component on the other hand has no formal responsibility to ensure safety. However, the decision component still needs to consider safety aspects in order for the vehicle to function properly. If the decision component makes decisions which are not considered safe the safety component will alter the decisions. This means that there is no guarantee that the vehicle will drive comfortably or even in the correct direction. In a perfect world the safety component should never have to alter a decision and only exist in order to satisfy safety certifications or ASIL requirements. Even in a world which is not perfect the safety component should not have to alter decisions frequently. If the decision component in a vehicle repeatedly makes decisions which are not allowed, the issue should be investigated and fixed. This concept is similar to what is discussed by Behere and Törngren [14], that a sufficiently fast decision component should be able to avoid threats on its own.

Developing two components which both take similar safety aspects into account, albeit to different degrees, could be considered unnecessary. However, it is important to remember the goal of the design. The proposed approach is not necessarily easier or faster to develop initially. The benefit comes later in the life cycle as updating functionality, which is not strictly safety related, requires significantly less testing. However, as mentioned in section *6.4 Evaluation of Results*, the problem with double implementation could be reduced if the decision component make use of the safe actions provided by the safety component.

As described in section *6.1 Requirements* there is a trade-off between performance and precision. One aspect to consider is how much the calculations can be simplified. Instead of performing resource heavy calculations on how the vehicle will move it might be possible to simplify and add larger margins to compensate for the loss of precision. However, larger margins would result in more actions being considered unsafe. A careful decision must be made of how much can be simplified while maintaining a large safeset. The decision must also take into account how much resources the calculations are allowed to use. In some cases it might be more appropriate to use more advanced calculations at the cost of extra resources.

It is necessary to decide where to draw the line between safety and functionality. In some cases it is obvious, for example it is the safety component's responsibility that the vehicle stops before a stationary vehicle but it is the decision component's responsibility that the vehicle stops smoothly to avoid passengers being uncomfortable. Other cases could be harder to decide, for example is it safety related to follow traffic rules? Deciding exactly what is safety related could be a hard problem and is not necessarily something which vehicle manufacturers will be allowed to decide on their own.

If the system is expected to only release safety-critical updates, the proposed solution might not be advantageous. In the event of such updates, the system has to be recertified and the separation of safety and functionality does not provide any benefits. However, it should be noted that updates which could seem safety-critical at first glance might not actually be safety-critical. For example, increasing fuel efficiency for the ego vehicle through smarter distance keeping to the vehicle in front is not safety-critical if the distance remains longer than the minimum safe distance.

7.2 Assumptions

As mentioned in section *6.2.1 Prediction*, assumptions increase the amount of actions which are considered safe. An alternative approach would be to consider all actions safe and add assumptions which specify when an action is not considered safe. In this study the first approach was chosen as it is a more cautious approach which reduces the risk from missing an assumption.

Even though a human driver makes assumptions on how other drivers will behave, it could be hard to prove that the same assumptions are safe for autonomous vehicles to make. If the safety component is only allowed to make weak assumptions, the vehicle will not be able to efficiently travel to its destination. The vehicle will be safe, but at the cost of longer transport times. However, regardless of whether safety constraints and functionality are split or combined, the system will only legally be allowed to make use of the same assumptions.

While assumptions could be considered hard to define, it can be argued that all systems use assumptions either implicitly or explicitly. It does not matter whether a system uses assumptions as a concept or not, if it uses any rule there is an implicit assumption behind it. For example, if the vehicle is allowed to drive forward in a certain context it is assumed that there is nothing which can appear in front of the vehicle.

7.3 Single Point of Failure

Section *5.3.3 Redundancy* suggests an approach for making the proposed architecture fail-operational through hardware redundancy. The suggested approach includes redundancy for *SafesetCalculator* and *ActionSelector*, see Figure 5.6, and uses two different approaches similar to what is suggested by Schnellbach et al [25]. One of the approaches is to use to triple redundancy with voting, as used by *ActionSelector* with voting in *VehicleControlCommunication*. The other approach suggested by Schnellbach et al. has the benefit of only requiring double redundancy, but can only be used when it is possible to check an answer for correctness, see section *4.4 Faults and Redundancy*. This approach is used when calculating a safeset by *SafesetCalculator*. While it is not possible to check a single safeset for correctness, the used approach is to consider the intersection of the safesets as correct. This approach works well if parts of a safeset contain faults, but not if a whole safeset is erroneous. If the process fails completely it would result in an empty intersection triggering an emergency action, which most likely would be emergency brake. The question which needs to be answered is whether an emergency action is safe enough in the event of a *SafesetCalculator* failing completely. While this study assumes that it is safe enough, the real answer will have to be investigated in another study. If it is not considered safe enough it would be necessary to use triple redundancy for *SafesetCalculator* as well.

This study proposes an approach for how redundancy can be implemented for *SafesetCalculator* and *ActionSelector*, but it does not specify how to avoid single point of failure when it comes to the communication between high-level components. In the proposed architecture, see Figure 5.3, this refers to *DecisionCommunication* and *VehicleControlCommunication*. However, since *DecisionCommunication* is not safety-critical it could be acceptable without redundancy in that case. *VehicleControlCommunication* has two tasks which are both safety-critical: to handle the vote between selected actions and to handle all communication between the safety component and the vehicle control component. From a safety component perspective this problem could be solved by having multiple *VehicleControlCommunication* components. This would require the vehicle control component to be able to receive multiple signals and vote between them. However, as written in section *1.4 Scope* this study only addresses problems which are created as a result of the division of safety constraints and functionality. Without the safety component the problem would exist between the decision component and the vehicle control component instead, thus this problem is left without a specified solution.

The proposed approach uses a monitor component, which can be seen in Figure 5.3 and is described in section *5.3.4 Monitor*. *Monitor*, similarly to the communications components, creates problems from being a single point of failure. As the monitor has the power to restart other components it can be problematic if the monitor fails and continuously restarts all other components. While one solution could be to use triple redundancy, similarly to one of the approaches discussed by Schnellbach et al. [25], it would just move the single point of failure to a voter. For small failures it could work if each component listened to three monitors and voted themselves, but it would be problematic if a component crashes completely. If a component fails and shuts down it cannot restart itself which means that another component needs to be able to restart it. However, the problem of which component should be able to restart other components is considered independent from the issue of separating safety constraints from functionality and has not been included in the study.

7.4 Alternative Applications

An added benefit of developing a safety component which is independent of the decision component is that it potentially could be used for manual drivers as well. In that case the driver would replace the decision component and the safety component would limit the driver if the driving is considered unsafe. However, developing a safety component which a manual driver does not consider too restricting could prove difficult and will have to be investigated in another study. A more likely approach might be that parts of the safety component could double as active safety measures or driver assistance for manual drivers.

As evident from section *6.4 Evaluation of Results* a potential application of the proposed approach is to use it for identification of safety constraints which limits the vehicle from driving as desired. For example, why a lane change is not considered safe in a certain circumstance. This information could be used to focus development on relevant areas. The areas could vary significantly, examples include: precision of sensors, speed of algorithms or which assumptions regarding other vehicles that safely can be made.

Potentially, the proposed approach could be applied in other types of autonomous vehicles as well. While the approach is designed with road vehicles in mind the approach is not inherently bound to that context. Obviously, the implementation and assumptions in other contexts would be vastly different, but the architecture could potentially be similar. Furthermore, it is possible that the approach could be used in other types of safety-critical systems as well. It should be noted that the evaluation of the architecture and studied literature is within the context of autonomous road vehicles, thus the uncertainty regarding applicability quickly increases when diverging from autonomous road vehicles.

7.5 Alternative Solutions

This study does not claim to have found the only solution to the studied problem, but rather it proposes one solution suitable for autonomous road vehicles. One alternative approach for decision making is to use deep learning, which can excel at finding structure in complex problems with large sets of data [35]. However, there exist a number of problems regarding the verification of such a solution [36]. Seshia et al. [36] propose some techniques which potentially could ease the verification process, but they do not make any claim to have solved the problem. A problem is to decide at which point the vehicle is trained for all possible situations. There can also be problems if the context changes and new situations arise. As the behavior is learned from data, rather than being written by human engineers [35], it can be problematic to manually adapt to a changed traffic rule.

If all those problems could be solved, and deep learning algorithms provided good decision making while being certifiable, there would be no need for a safety component similar to what is proposed in this study. However, the safety component could be used to allow deep learning in an uncertified decision component. This scenario would make use of deep learning's problem solving capabilities while avoiding certification issues.

Removing the need to certify the decision component is done by having something else guarantee the safety aspect. In the proposed solution this is done through run-time verification by the safety component. An alternative could be to verify the safety of the system at compile-time, with the benefit of requiring less hardware resources when deployed in production. However, formal verification at compile-time of this type of system would result in a near infinite state problem [37].

There exist similar solutions on how to handle safety in related domains. In the domain of autonomous air force flight systems, Hinchman et al. [37] identified issues with compile-time certification as the aircraft could encounter unanticipated elements or actions when in use. Due to the importance of having aircrafts which can handle real time changes in the environment and mission detail, run-time verification was deemed a better choice. The run-time verification presented by Hinchman et al. generates a set of collision avoidance paths which are deemed safe. As long as the proposed action can lead to one of the paths, it can be safely executed by the aircraft. One major difference to the proposed solution in this study is that the solution described by Hinchman et al. generates potential paths according to the vehicle's limitations at compile-time, while this study's solution calculates them at run-time using data from the vehicle control component. The advantage of having them generated at compile-time is that less resources are used at run-time, which means that more paths can be tested. However, a major disadvantage is that the paths might be wrong if the capabilities change during operation, for example due to the vehicle carrying heavy cargo. This would be a problem in a road context as the friction between the road and the tires can vary, and thus the vehicle might not be able to follow the same paths. It should also be mentioned that Hinchman et al. do not present how a path is actually checked for whether it is safe.

To ensure safety in small airplanes, Hook et al. [38] describe a method where a simpler system monitors the more complex autopilot system. The simpler system takes the burden of safety certification and ensures the autopilot does not execute unsafe action. As the airplane is not fully autonomous, control is given back to the pilot in the event that the autopilot tries to execute an unsafe action. However, for a completely autonomous road vehicle, such a solution would not be possible as there is no driver to take control.

Phan et al. [39] present a solution for mobile robots. In their solution, a complex decision component controls the robot during normal operation. However, it can be difficult to verify that a complex decision component fulfills all safety requirements. By having a more basic decision component take over as soon as a hazard is identified, it is not necessary to assure the required safety levels for the more complex component. The difference between the solution by Phan et al. and the solution proposed in this study is that they do not verify actions and instead override as soon as the robot enters a context with potential hazards. This would not be feasible for a road vehicle as it would be in contexts with potential hazards in a lot of scenarios and thus the complex decision component would be constantly overruled. To ensure contexts with potential danger are correctly identified for a road vehicle, a much more advanced monitor than they propose would have to be implemented.

Guiochet et al. [40] present a survey on how safety-critical robots manage safety. From the result of the survey, Guiochet et al. note that a popular approach is to use a monitor which forces the robot into a recovery state when a hazard is detected, similarly to the approach presented by Phan et al. [39]. Guiochet et al. briefly present a study by Woodman et al. [41] which filter actuator requests, or stop the robot, if safety is uncertain or undefined for a specific action. Similarly to the solution proposed in this study, a component is added between the decision component and their version of the vehicle control component [41]. If a decision could lead to a fault or other danger, then it is prevented from being executed on the robot. However, Woodman et al. do not say what would be the result of not executing a decision, or if the decision component is notified.

7.6 Limitations

This section discusses validity threats and limitations to the study. While the limitations are described by the scope of the study, the researchers want to highlight two limitations which could easily be forgotten while considering the result. The validity threats are categorized according to the methodology proposed by Feldt and Magazinius [42]. While there is no apparent conclusion or construct validity threat, there is an uncertainty whether the proposed separation is acceptable from a legal point of view, meaning the certification will still have to be done on the whole system configuration.

The study was performed as action research, meaning the researchers were the ones designing and implementing the architecture as well as compiling the results of the evaluation. This leads to a confirmability threat as the researchers could already have an established perception of the result. However, this risk was minimized by including industry practitioners in both the design and evaluation phases. Although, this involvement creates a credibility risk as continuous involvement from industry practitioners to a certain degree means that they evaluated the result of some of their own ideas. This risk is not considered large as some of the practitioners initially did not believe the separation would create feasible solution, thus potential bias would most likely be towards a more negative evaluation.

During the literature review four different databases were searched for papers about autonomous road vehicles. Had additional databases been used more papers would most likely have been found. Furthermore, the search query and the inclusion criteria meant that papers which could have been relevant, but not specific to autonomous road vehicles, were not found. If more resources had been available for the literature review, papers on a more general topic could have been included as well. This leads to a dependability threat as a more thorough literature review could have yielded more patterns to consider when designing the architecture.

The architecture, and the proposed approach, are meant to be applicable in autonomous road vehicles in general. But there is an external validity threat as the architecture was only evaluated through application at one company and for one type of vehicle. As a result, the architecture might not work as well when being applied in other systems. Furthermore, the prototype was only tested in highway cases, meaning the solution might not be suitable for other traffic environments, such as intersections. However, while this is the case for the prototype implementation, the architecture was designed without a specific company or vehicle in mind and should be applicable in any other autonomous road vehicle as well.

It is important to be aware of what kind of problems the proposed architecture is trying to solve. There are several problems not considered in this study which needs to be addressed when designing an autonomous road vehicle. This study only considers problems related to how decision making could be separated into safety constraints and other functionality. This means that problems related to how to make good driving decisions is not considered in this report even though it needs to be solved when designing an autonomous road vehicle.

Another aspect to consider is that the actual safety requirements for an autonomous road vehicle are assumed to be known and correct. The problem of deciding how to define a safe and unsafe action is not a problem this study tries to solve. The same goes for how to translate a definition of safe and unsafe actions into concrete requirements. The study uses a simple and high-level definition, however, this definition should not to mistaken for a final answer.

8

Conclusion

The study has gathered information from existing literature regarding which software architectural patterns and tactics are currently used for safety in autonomous road vehicles. The literature review did not find any papers in this domain which present how a separation of safety constraints and functionality could be performed in practice. However, it revealed software architectural patterns for solving a number of safety related problems. Using these patterns, the study proposes a software architecture which separates the responsibility for safety constraints and functionality. Without responsibility for safety requirements, the testing procedure for the decision component could be significantly faster and cheaper.

Through application in an industrial case it became clear that the proposed solution is a feasible approach for the studied problem. It could also be seen that making correct assumptions regarding the behavior of other traffic participants is critical, and one of the most difficult problems to solve.

It is important to remember that the division of responsibilities does not mean that the decision component should ignore safety aspects. If the decision component proposes unsafe actions then the safety component will override the actions, which results in the vehicle not attempting to drive comfortably or reach its destination. Thus it is important that the decision component either asks the safety component which actions are safe, and act accordingly, or calculates actions based on safety constraints itself.

The primary goal of the proposed architecture is to avoid recertification of safety requirements in case of functionality updates. However, there also exists an alternative application where the safety component could be used as a tool to show which limitations currently are stopping the vehicle from operating as desired. Thus the safety component could also facilitate the development process of autonomous vehicles.

8.1 Contribution

The findings of the study could help with developing autonomous road vehicles systems which allow frequent updates. Meaning each update should not require a restart of the testing and certification procedure. This could potentially result in both faster and cheaper releases of updated software while keeping the same level of strictness for safety.

The proposed approach could also be used to show which safety aspect that currently is limiting vehicles from driving as desired. It could show both whether new functionality is allowed by the current safety constraints and how much an improvement in a safety related area would increase the amount of allowed actions.

Aside from the study's purpose, the study contributes to the software engineering body of knowledge when it comes to the applicability of different software architecture patterns and tactics in certain contexts. Furthermore, the study contributes with knowledge regarding the design of systems where safety requirements should be independent from the rest of the requirements.

For potential future studies, this study has provided a literature review of software architectural patterns and tactics used in relation to safety in autonomous road vehicles. While it cannot be claimed the review is exhaustive, it could be used as a base for studies in a similar area.

8.2 Future Work

The general approach suggested in this study has only been tested in one context. Before the approach can be claimed to be completely generalizable it should be tested for other vehicles, preferably with different software infrastructure. The approach should also be tested on a physical vehicle and not only in simulations. Furthermore, it could be investigated whether the same or a similar approach could be used in other types of safety-critical systems.

It should not be forgotten that this study only addresses the separation of safety and functionality. More research is still needed regarding actual features necessary in an autonomous road vehicle. This applies both to features for the safety component and the decision component. There could also be other ways to perform the separation which would require less hardware resources or result in more precise, and thus less limiting, safesets.

Bibliography

[1] J. C. Knight, “Safety critical systems: challenges and directions,” in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, May 2002, pp. 547–550.

[2] S. M. Veres, L. Molnar, N. K. Lincoln, and C. P. Morice, “Autonomous vehicle control systems—a review of decision making,” *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 225, no. 2, pp. 155–195, 2011.

[3] *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*, SAE International J3016, 2014.

[4] K. K. Aggarwal, Y. Singh, and J. K. Chhabra, “An integrated measure of software maintainability,” in *Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No.02CH37318)*, 2002, pp. 235–241.

[5] H. K. N. Leung and L. White, “Insights into regression testing [software testing],” in *Proceedings. Conference on Software Maintenance - 1989*, Oct 1989, pp. 60–69.

[6] S. Nair, J. L. de la Vara, M. Sabetzadeh, and L. Briand, “An extended systematic literature review on provision of evidence for safety certification,” *Information and Software Technology*, vol. 56, no. 7, pp. 689 – 717, 2014.

[7] E. Bringmann and A. Krämer, “Model-based testing of automotive systems,” in *2008 1st International Conference on Software Testing, Verification, and Validation*, April 2008, pp. 485–493.

[8] M. Campbell, M. Egerstedt, J. P. How, and R. M. Murray, “Autonomous driving in urban environments: approaches, lessons and challenges,” *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1928, pp. 4649–4672, 2010.

[9] R. L. Nord, I. Ozkaya, R. S. Sangwan, and R. J. Koontz, “Architectural dependency analysis to understand rework costs for safety-critical systems,” in

Companion Proceedings of the 36th International Conference on Software Engineering, ser. ICSE Companion 2014. New York, NY, USA: ACM, 2014, pp. 185–194.

[10] T. Kelly, "Using software architecture techniques to support the modular certification of safety-critical systems," in *Proceedings of the Eleventh Australian Workshop on Safety Critical Systems and Software - Volume 69*, ser. SCS '06. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 53–65.

[11] S. Behere, F. Asplund, A. Söderberg, and M. Törngren, *Architecture Challenges for Intelligent Autonomous Machines.* Cham: Springer International Publishing, 2016, pp. 1669–1681.

[12] S. Underwood, D. Bartz, A. Kade, and M. Crawford, *Truck Automation: Testing and Trusting the Virtual Driver.* Cham: Springer International Publishing, 2016, pp. 91–109. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-40503-2_8

[13] Y. Liu and U. Özgüner, "Human driver model and driver decision making for intersection driving," in *2007 IEEE Intelligent Vehicles Symposium*, June 2007, pp. 642–647.

[14] S. Behere and M. Törngren, "A functional reference architecture for autonomous driving," *Information and Software Technology*, vol. 73, pp. 136 – 150, 2016.

[15] T. Luettel, M. Himmelsbach, and H. J. Wuensche, "Autonomous ground vehicles – concepts and a path to the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1831–1839, May 2012.

[16] *Road vehicles - Functional safety, ASIL-oriented and Safety-oriented Analyses*, ISO 26 262-9, 2011.

[17] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.

[18] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, and W. Wood, "Attribute-driven design (ADD), version 2.0," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2006-TR-023, 2006.

[19] R.L. Baskerville and A.T. Wood-Harper, "A critical perspective on action research as a method for information systems research", *Journal of Information Technology*, vol. 11, pp. 235-246, 1996.

[20] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, Jul 1999.

[21] E. T. McGee and J. D. McGregor, "Using dynamic adaptive systems in safety-critical domains," in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.* ACM, 2016, pp. 115–121.

[22] J. M. Borrmann, F. Haxel, A. Viehl, O. Bringmann, and W. Rosenstiel, "Safe and efficient runtime resource management in heterogeneous systems for automated driving," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Sept 2015, pp. 353–360.

[23] M. Hörwick and K. H. Siedersberger, "Strategy and architecture of a safety concept for fully automatic and autonomous driving assistance systems," in *2010 IEEE Intelligent Vehicles Symposium*, June 2010, pp. 955–960.

[24] G. Brancovici, "Towards trustworthy intelligence on the road: A flexible architecture for safe, adaptive, autonomous applications," in *2007 IEEE Congress on Evolutionary Computation*, Sept 2007, pp. 4230–4237.

[25] A. Schnellbach, M. Hirz, and J. Fabian, "Comparison of fail-operational software architectures from the viewpoint of an automotive application," *e & i Elektrotechnik und Informationstechnik*, vol. 133, no. 6, pp. 283–293, 2016.

[26] J. Kim, G. Bhatia, R. Rajkumar, and M. Jochim, "Safer: System-level architecture for failure evasion in real-time applications," in *2012 IEEE 33rd Real-Time Systems Symposium*, Dec 2012, pp. 227–236.

[27] F. W. Rauskolb, K. Berger, C. Lipski, M. Magnor, K. Cornelsen, J. Effertz, T. Form, F. Graefe, S. Ohl, W. Schumacher *et al.*, "Caroline: An autonomously driving vehicle for urban environments," *Journal of Field Robotics*, vol. 25, no. 9, pp. 674–724, 2008.

[28] E. Jakobsson, A. Beutner, S. Pettersson, A. Bartels, F. Seglö, A. Lindqvist, A. Nilsson, H. Stratil, F. Ahlers, K. Fuerstenberg, A. Amditis, G. Thomaidis, K. Pagle, F. Flemisch, G. Temme, H. Mosebach, A. Schieben, J. Schomerus, E. Frey, M. Sabatier, N. Rauch, P. Luithardt, M. Armbruster, A. Fortelle, F. Nashashibi, L. Nouveliere, S. Mammar, S. Glaser, A. Dufaux, D. Manetti, P. Allemann, S. Boverie, A. Giralt, M. Strauss, W. Schrinner, A. Abele, H. Zeng, M. Fiedler, and A. Hoess. The future of driving, deliverable d12.1 architecture. [Online]. Available: www.haveit-eu.org

[29] M. Werling, M. Goebl, O. Pink, and C. Stiller, "A hardware and software framework for cognitive automobiles," in *2008 IEEE Intelligent Vehicles Symposium*,

June 2008, pp. 1080–1085.

[30] M. Goebl, M. Althoff, M. Buss, G. Farber, F. Hecker, B. Heissing, S. Kraus, R. Nagel, F. P. Leon, F. Rattei, M. Russ, M. Schweitzer, M. Thuy, C. Wang, and H. J. Wuensche, "Design and capabilities of the munich cognitive automobile," in *2008 IEEE Intelligent Vehicles Symposium*, June 2008, pp. 1101–1107.

[31] T. Gindele, D. Jagszent, B. Pitzer, and R. Dillmann, "Design of the planner of team AnnieWAY's autonomous vehicle used in the DARPA Urban Challenge 2007," in *2008 IEEE Intelligent Vehicles Symposium*, June 2008, pp. 1131–1136.

[32] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun, "Junior: The stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[33] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, E. Kaus, R. G. Herrtwich, C. Rabe, D. Pfeiffer, F. Lindner, F. Stein, F. Erbs, M. Enzweiler, C. Knoppel, J. Hipp, M. Haueis, M. Trepte, C. Brenk, A. Tamke, M. Ghanaat, M. Braun, A. Joos, H. Fritz, H. Mock, M. Hein, and E. Zeeb, "Making bertha drive - an autonomous journey on a historic route," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, Summer 2014.

[34] *Road Vehicles - Heavy Commercial Vehicle Combinations and Articulated Buses - Lateral Stability Test Methods*, ISO 14 791, 2002.

[35] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[36] S. A. Seshia, D. Sadigh, and S. S. Sastry, "Towards verified artificial intelligence," *arXiv preprint arXiv:1606.08514*, 2016.

[37] J. Hinchman, M. Clark, J. Hoffman, B. Hulbert, and C. Snyder, "Towards safety assurance of trusted autonomy in air force flight critical systems," in *Computer Security Applications Conference, Layered Assurance Workshop*, 2012.

[38] L. R. Hook, M. Clark, D. Sizoo, M. A. Skoog, and J. Brady, "Certification strategies using run-time safety assurance for part 23 autopilot systems," in *2016 IEEE Aerospace Conference*, March 2016, pp. 1–10.

[39] D. Phan, J. Yang, D. Ratasich, R. Grosu, S. A. Smolka, and S. D. Stoller, *Collision Avoidance for Mobile Robots with Limited Sensing and Limited Information About the Environment.* Cham: Springer

International Publishing, 2015, pp. 201–215. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-23820-3_13

[40] J. Guiochet, M. Machin, and H. Waeselynck, “Safety-critical advanced robots: A survey,” *Robotics and Autonomous Systems*, vol. 94, pp. 43 – 52, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889016300768

[41] R. Woodman, A. F. Winfield, C. Harper, and M. Fraser, “Building safer robots: Safety driven control,” *The International Journal of Robotics Research*, vol. 31, no. 13, pp. 1603–1626, 2012.

[42] R. Feldt and A. Magazinius, “Validity threats in empirical software engineering research-an initial survey.” in *SEKE*, 2010, pp. 374–379.

A

Highway Scenarios

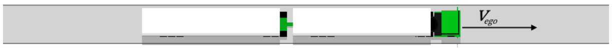


Figure A.1: Scenario 1: One lane without roadsides and with no surrounding vehicles.

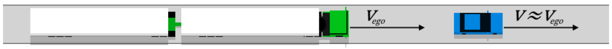

V_{ego}
$V \approx V_{ego}$

Figure A.2: Scenario 2: One lane without roadsides and with a vehicle in front of the ego vehicle

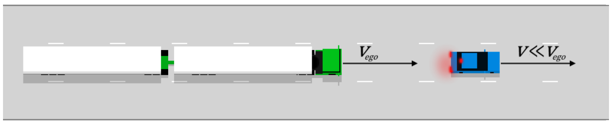

V_{ego}
$V \ll V_{ego}$

Figure A.3: Scenario 3: Multiple lanes with a vehicle braking in front of the ego vehicle.

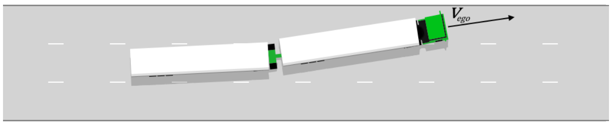


Figure A.4: Scenario 4: Multiple lanes and no other vehicles. The ego vehicle performs a lane change.

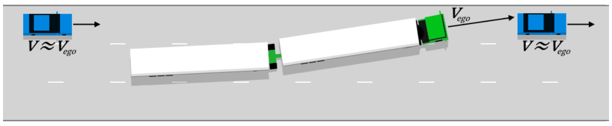

$V \approx V_{ego}$
V_{ego}
$V \approx V_{ego}$

Figure A.5: Scenario 5: Multiple lanes, the ego vehicle attempts to change into a lane where there are vehicles behind and in front of the ego vehicle.

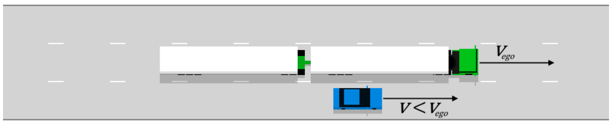


Figure A.6: Scenario 6: Multiple lanes, the ego vehicle attempts to overtake a vehicle in another lane.

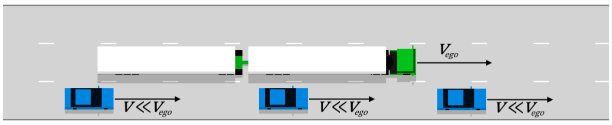

V_{ego}
$V \ll V_{ego}$
$V \ll V_{ego}$
$V \ll V_{ego}$

Figure A.7: Scenario 7: Multiple lanes where the traffic in a lane next to the ego vehicle's lane is dense and moves slowly.

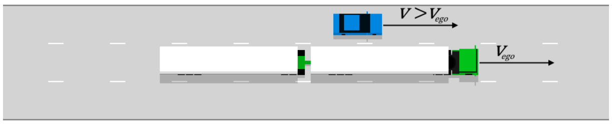


Figure A.8: Scenario 8: Multiple lanes where the ego vehicle is being overtaken by a vehicle in another lane.

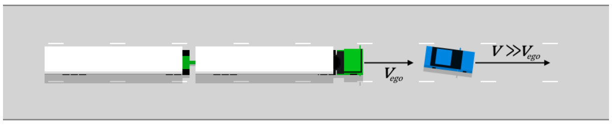

V_{ego}
$V \gg V_{ego}$

Figure A.9: Scenario 9: Multiple lanes where a vehicle changes into the ego vehicle's lane shortly after overtaking the ego vehicle.

B

Industrial Case Requirements

B.1 Context Requirements

ID: C1
Title: Support different traffic situations
Description: The safety component should be able to function in all different traffic situations where the vehicle is supposed to be used.
Dependency: -

ID: C2
Title: Independent of specific vehicle configurations.
Description: The safety component should function properly independently of the vehicle it is used in. This includes that the component should be able to handle vehicles of different size, weight, maneuver capabilities.
Dependency: -

ID: C3
Title: Assumptions on environment
Description: The safety component should be able to make assumptions regarding the behavior of different traffic participants.
Dependency: -

ID: C4
Title: Flexible assumptions
Description: The safety component should be able to change between different sets of assumptions while running. This can be used to change assumptions on different kinds of roads or when driving in countries with different traffic rules.
Dependency: C3

B.2 Safety Requirements

ID: S1
Title: Responsibility for actions
Description: The safety component should be responsible for each action. All actions should be approved or changed by the safety component before reaching vehicle control.
Dependency: -

ID: S2
Title: Allow safe actions
Description: The safety component should allow actions which it considers safe. The safety component is not allowed to change an action it considers safe. The safety component should minimize the number of false negatives, i.e. avoid disallowing actions which in reality are safe.
Dependency: S1

ID: S3
Title: Modify unsafe actions
Description: The safety component should modify all incoming actions which it considers unsafe. The modified action should be the closest action which is considered safe. When determining the closest action the closest steering angle will be chosen.
Dependency: S1

ID: S4
Title: Handle absence of action
Description: The safety component should be able to safely maneuver the vehicle in the absence of actions from other components. Absence of actions should lead to a safe stop.
Dependency: S1

ID: S5
Title: Not be the cause of an accident
Description: The safety component should ensure that the vehicle is not the cause of an accident. The safety component should not allow the vehicle self-inflict a state where it cannot safely stop.
Dependency: S1, S2, S3, S4

ID: S6
Title: Try to safely avoid accidents
Description: The safety component should avoid all accidents, including when the vehicle itself is not at fault. If external factors force the vehicle into an unsafe state the safety component should only allow actions which minimizes the risk.
Dependency: S1, S2, S3, S4, S5

ID: S7
Title: Provide safe actions
Description: The safety component should be able to provide information for other components about which actions are considered safe.
Dependency: -

B.3 Performance Requirements

ID: P1
Title: Real-time execution
Description: The safety component should be able to run in real-time.
Dependency: -

ID: P2
Title: Low delay
Description: The safety component should minimize the time it takes to make a decision after it has received an action from the decision component.
Dependency: -

ID: P3
Title: Memory usage
Description: The safety component should minimize the memory usage.
Dependency: -

ID: P4
Title: CPU usage
Description: The safety component should minimize the processing power used.
Dependency: -

ID: P5
Title: Number of CPUs
Description: The safety component should minimize the number of CPUs that have to be used.
Dependency: -

ID: P6
Title: Limited communication
Description: The safety component should minimize the amount and size of messages it sends between components.
Dependency: -

B.4 Architecture Requirements

ID: A1
Title: Low external ASIL
Description: The safety component should not increase the ASIL requirements for surrounding components.
Dependency: -

ID: A2
Title: Low internal ASIL
Description: The safety component should minimize the ASIL requirements for its subcomponents.
Dependency: -

ID: A3
Title: No ASIL on decision component
Description: The decision component should have ASIL QM, i.e. no ASIL requirements.
Dependency: -

ID: A4
Title: Avoid cascading tests
Description: A change in the decision component should not require testing of the safety component.
Dependency: A3