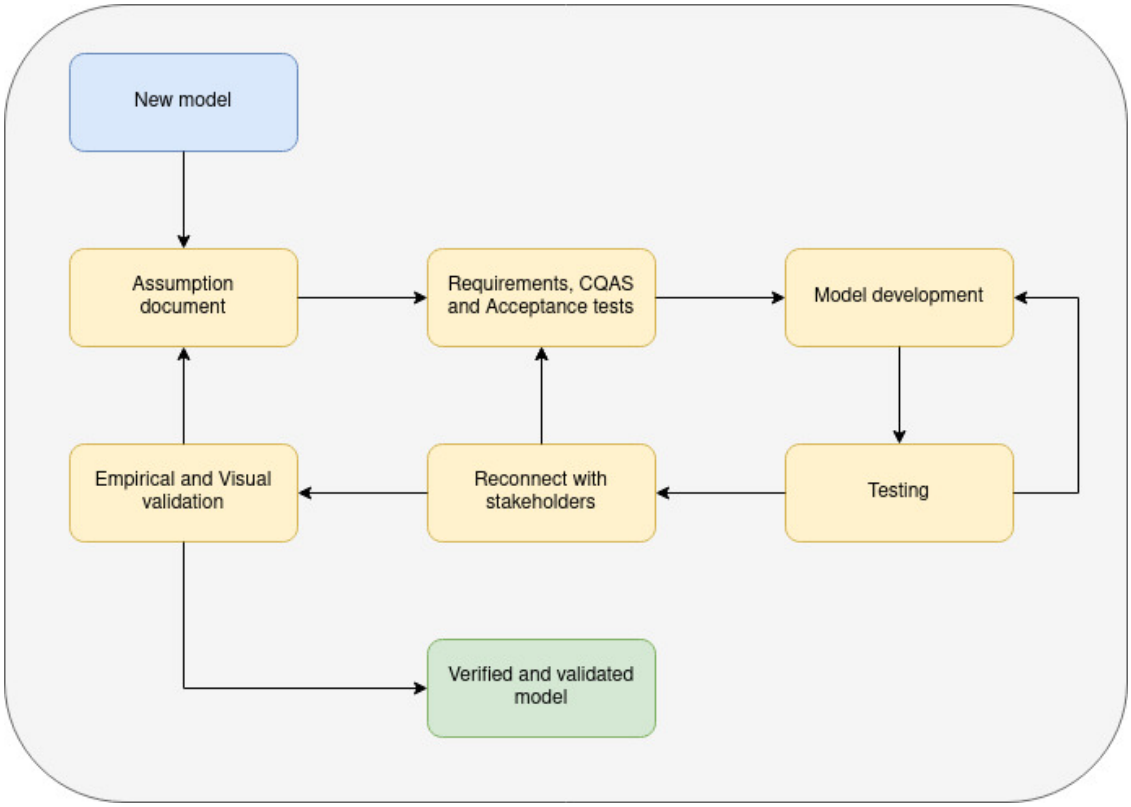
CHALMERS
UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG



A Framework for Verification and Validation of Simulation Models in esmini

With an Example of Framework Application

Master's thesis in Computer science and engineering

Daniel Olsson, Eric Rylander

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

A Framework for Verification and Validation of Simulation Models in esmini

With an Example of Framework Application

Daniel Olsson, Eric Rylander

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

A Framework for Verification and Validation of Simulation Models in esmini
With an Example of Framework Application
Daniel Olsson, Eric Rylander

Supervisor: Sam Jobara, Department of Computer Science and Engineering
Advisor: Mikael Andersson, Volvo Car Group
Examiner: Robert Feldt, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A simplified flowchart displaying the primary steps of the framework.

Typeset in LaTeX
Gothenburg, Sweden 2022

A Framework for Verification and Validation of Simulation Models in esmini
With an Example of Framework Application
Daniel Olsson, Eric Rylander
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

As the use of simulations is becoming more prominent in the industry, the need for verified and validated simulation models becomes more important. For the simulator esmini, there are no concrete methods or techniques for the V&V process, which this thesis aims to unravel. This thesis presents a verification and validation (V&V) framework for the traffic simulator esmini, with the purpose of providing a foundation for the V&V process and contribute to the research field of simulation validation. The framework was developed by performing desk research on existing V&V methodologies and combining found techniques with software engineering concepts such as quality attribute scenarios (QAS) and acceptance test-driven development (ATDD). The research was done in collaboration with Volvo Cars. Simultaneously, a lane independent routing model for esmini was developed to solve a case provided by Volvo Cars, to which the framework was applied in order to provide an example of how the framework could be applied to a simulation model and to evaluate the framework in terms of usefulness and effectiveness. The resulting framework consisted of six iterative steps: creating an assumptions document, defining requirements and acceptance tests, developing the model, testing and refactoring, reconnecting with stakeholders, and performing empirical and visual validation. The fundamental concept of the framework is to perform the steps iteratively, in order to continuously improve the simulation model. The evaluation of the framework was primarily done through fault injection and developer feedback from Volvo Cars which were encouraging. Overall, the conclusion is that the framework provides necessary guidance for the V&V process in esmini and that it could be used for V&V in other similar simulators.

Keywords: V&V, Verification, Validation, Framework, esmini, OpenDRIVE, OpenSCENARIO, Simulation models

Acknowledgements

We would like to thank our supervisor, Sam Jobara, for providing support and feedback throughout the thesis. We would also like to thank our advisor at Volvo Cars, Mikael Andersson, for taking the time to help us understand the case and esmini, and providing helpful insights on our thesis. We would also like to thank Emil Knabe, the main developer on esmini, for helping us to locate and fix bugs in our model.

Daniel Olsson & Eric Rylander, Gothenburg, June 2022

Acronyms

ABS	Agent-Based Simulation
API	Application Programming Interface
ASAM	Association for Standardization of Automation and Measuring System
ATDD	Acceptance Test-Driven Development
CQAS	Concrete Quality Attribute Scenario
DES	Discrete-Event Simulation
DSRP	Design Science Research Process
esmini	Environment Simulator Minimalistic
LIRM	Lane Independent Routing Model
MBT	Model-Based Testing
OEI	Observable emerging information
PBT	Property-Based Testing
QAS	Quality Attribute Scenario
V&V	Verification and Validation

Contents

List of Figures

List of Tables

1
Introduction

Verification and validation (V&V) are two procedures that software engineers should use to increase the correctness, accuracy, and reliability of developed software products. Verification is the procedure to determine if a developed software represents the conceptual design and specifications [1], [2], while validation is the procedure to determine if the behavior of the developed software fulfills a set of criteria for the intended use [2]. In other words, verification answers the question "have we built the software right?" while validation answers the question "have we built the right software?" [2]. V&V are even more important for simulation models, as these procedures should be used to determine the correctness of the simulated models' data compared to a realistic scenario.

The aim of the study is to create a framework for applying V&V to simulation models developed in Environment Simulator Minimalistic (esmini) at Volvo Cars. The framework should be presented as an informal process containing a set of systematic procedures applied in given steps. To create the framework, the plan is to define new methods, but also take ideas or advantages from existing framework techniques to improve on our framework. It should define the methods needed to assess both verification and validation of a simulation model. The framework is intended to be tested during the study by using it for the V&V process on a newly developed simulation model, in order to evaluate the framework's usability and effectiveness. The goal of the developed framework is to provide a set of procedures for V&V of future models, as well as identify problems with current simulation models.

1.1 Statement of the Problem

Verification and Validation: From what was found, there were no set frameworks for V&V in regard to the development of models for esmini, this could be seen in previous master thesis works [3], [4]. One of the theses [3] approaches validation of the developed simulation model by scenario testing, which states a number of given scenarios and discusses the behavior of the model in these scenarios. While the other thesis [4] briefly mentions validation of the model, no steps are given. To ensure that the future development of simulation models for esmini works as intended, a framework for V&V would be beneficial.

The found V&V frameworks and methodologies can not be directly applied to esmini as they are too general and loosely defined. Therefore, there is a need for a frame-

work in the field of V&V that fits esmini and can ensure that proper models are developed.

Simulation modeling: The area of computer simulation is related to modeling [5, p. 920], where a mathematical model is used by a simulation. Another definition for computer simulation is that executed computer code provides solutions to the equations of a model [5, p. 39]. Simulations enable computers to mimic the dynamical behavior of a real system under specified conditions. Sargent [6] emphasizes that a simulation model should have a distinct purpose in order for validation to be performed with regard to that purpose. Without a properly defined purpose of the simulation, the correctness and reliability of the targeted system will not be applicable for V&V. Furthermore, the absence of V&V in a simulation will result in insignificant outcomes achieved from the simulation models.

The importance of simulation is becoming more prominent and has become a common part of the testing process for various technical fields [7]. Simulation models need to be verified and validated in order to find model errors and reach the desired correctness and reliability.

Traffic simulation: Simulation of traffic scenarios require V&V to resemble real-life scenarios as close as possible or to test simplified scenarios such as those used by Euro NCAP [8]. Certification agencies and government authorities work together on developing maneuver descriptions to be used for testing, validation, and certification of vehicle safety systems and autonomous vehicles [9]. The simulation model's representativeness of real traffic scenarios is essential in order to apply results captured by the simulation in a similar traffic scenario with real environments and vehicles. Moreover, the importance of effective and efficient traffic simulation is connected to the safety of vehicles and pedestrians. An adequate V&V process used to evaluate a traffic scenario can benefit the safety of all actors involved. The Swedish National Road and Transport Research Institute (VTI) conducts research on human behavior in transport systems [10]. They use an in-house custom simulator environment to recreate realistic driving experiences, which makes it possible to perform traffic experiments and study various factors and their effects. Demonstrably, simulators are tools to improve traffic safety with respect to technologies, vehicle characteristics, road design and traffic environments among other things.

1.2 Purpose and Significance of Study

The purpose of this study is to create a framework for V&V for simulation modeling in esmini, in order to create a more accurate and reliable V&V process. In this case, the simulation models define road maneuver behavior inside the simulator. The framework is intended to benefit both researchers and practitioners that are working on traffic simulation modeling.

The thesis aims to improve existing modeling and simulation V&V by introducing a V&V framework for simulation model development in the simulator esmini.

The goal is that the developed framework can be used as a foundation for future V&V process development for esmini. It should also help improve simulation models and therefore, as mentioned in the statement of problems, help improve public safety through the increase of accurate simulation models. It will provide new aspects to the research on simulation modeling V&V.

1.3 Research Questions

The connections and steps between the research questions are visualized in a flowchart diagram in Figure 1.1.

RQ1: Develop and propose an optimized Validation & Verification framework for simulation models with the inclusion of lane changing scenarios for the esmini simulator.

- What steps are needed to create a V&V framework?
- What procedures are used in the framework?

Use knowledge from existing V&V frameworks and knowledge gathered from current simulation models in esmini to find and create procedures for the new framework. The developed V&V framework will consist of a set of systematic procedures and rule-based decisions, integrated to fulfill the functions of the new V&V framework.

RQ2: How can the created framework be applied to a simulation model containing road maneuver scenarios from the standard ASAM OpenSCENARIO? The application will be evaluated based on the terms of usefulness and effectiveness.

Document the application of the new framework on a developed simulation model containing road maneuver scenarios, and evaluate the framework based on a set of metrics measured during the application. The framework will be evaluated by performing simulated experiments, in the form of fault injections and removal of parts of the framework. Faults will be injected into the simulation model to verify that the faults are detected by some step of the framework. By removing parts of the framework, the usefulness and impact of the excluded steps will be presented.

1.4 Limitations and delimitations

The main limitations of the thesis are the time limit of approximately 5 months and that the effort needed to be adapted to what is possible for two people to complete during the duration of the thesis. Another important limitation of the framework that is to be developed is that it can not be considered as a guarantee for developing flawless simulation models, however it will improve the developed models with the improved V&V process.

To delimit the ASAM standards that were used in this thesis, only the most current versions of OpenSCENARIO (1.1.1) and OpenDRIVE (1.7.0) were considered. This

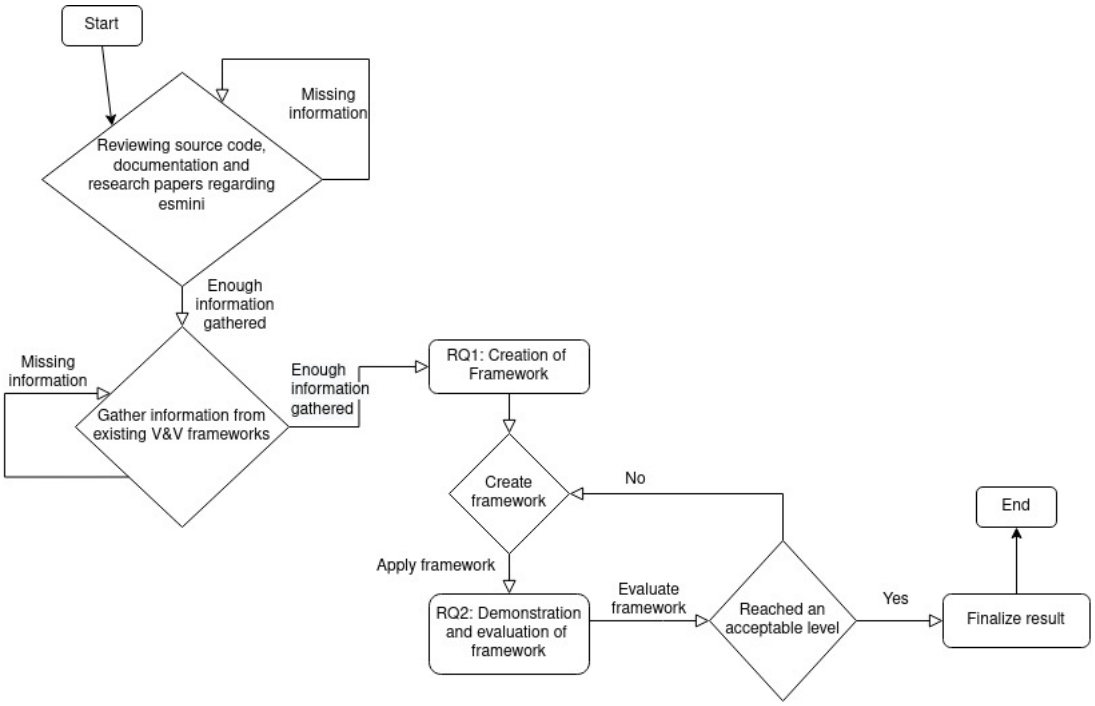


Figure 1.1: Flowchart of research questions.

delimitation was added to reduce the confusion and conflicting information that could occur if several versions were used, but also to limit the complexity of the development of both the framework and the model. Conduction of a complete literature review of known simulation model V&V methodologies and frameworks was not deemed feasible as it would take too much time, resulting in that the process of creating the framework and the application of it and model development might not have been completed in time. The choice of applying the framework on a singular model was as a delimitation due to the time limit of the thesis.

1.5 Thesis outline

The thesis is organized as follows: Chapter 2 provides an overview of the theoretical background and related works. In Chapter 3, the methods used in the thesis are presented and motivated. The developed framework is described in Chapter 4, while Chapter 5 provides an example of how the developed framework can be used on a model in esmini. Chapter 6 presents the results in regard to the research questions. The discussion is presented in Chapter 7 and conclusion can be seen in Chapter 8.

2
Theory

This chapter will present the related works, and introduce the needed theoretical background. Sections 2.1 and 2.2 present and summarize the related works, while the succeeding sections introduce important theory and concepts for the understanding of the thesis and framework.

2.1 Verification and validation in regard to simulation models

V&V is applied to the software process to ensure realistic and reasonable solutions [1]. This is increasingly important for simulation models, as users and individuals affected by the model need to know that the results are correct [6]. To validate a model, there should be a specific purpose defined so that the model can be validated with respect to the purpose. Riedmaier et al. [7] state that verification and validation must be performed on simulation models to assess the models' inherent errors and accuracy. Accurate models of reality have become the decisive factor for simulation credibility and trustworthiness.

Sargent [6] discusses and proposes different approaches for V&V of simulation models, where the different approaches have takes on who and how the V&V process should be performed. The first approach that Sargent discusses is that the development team responsible for the model gets to decide if the model has reached a valid state, through testing and evaluations made during the developing process. However, Sargent states that this approach usually should be avoided if possible. Instead, Sargent proposes that if the team size is small, it is better to involve the model users in the validation of a model. This approach moves the focus of determining validity from the developer to the user, which helps with model credibility. The third approach is the use of independent verification and validation, where a third party is used to decide if the model is valid. This approach can be done during different stages, which Sargent discusses, where the two main stages are concurrent with the development or after the development. This approach is also supported by the International Software Testing Qualifications Board (ISTQB) [11], as they state that testing should be performed at multiple levels where a part of those are performed by independent testers or test organizations. The last approach Sargent presents is through scoring, where a model is given scores based on the outcome of the validation steps performed, at the end of the validation the model's score determines if the model should be seen as valid. Sargent states that he does not believe

that this is an approach that should be used due to several reasons, one being that a model could still pass even if it is flawed in some way.

Sargent [6] also discusses four concepts that influence different phases through the validation process: data validity, conceptual model validity, computerized model verification and operational validity. Data validity is often not considered to be a part of model validation, as it is often difficult and time-consuming to obtain appropriate and accurate data. However, the data is needed for three main purposes, which are building the conceptual model, validating the model, and performing experiments. According to Sargent [6], building and validating the model are usually the reasons that data validity is an important part of model validation. Conceptual model validation is done on the conceptual model, where the theories and assumptions made for the model are determined to be represented correctly. The model's structure, logic, mathematics and causal relationship are evaluated based on if they are reasonable for the purpose of the model. Computerized model verification is done to ensure that the programming and implementation of the model are correct, the main purpose of this step is to ensure that the implementation is tested and deemed error-free. The last phase, operational validity, determines the validity of the simulation model's output. The output is evaluated based on the accuracy required for the intended purpose in the domain where the model is applied. It is during this phase that most of the validation and evaluation of the model takes place [6].

MITRE [2] discusses how the V&V process can be extended within certain safety critical fields, into the VV&A process. The extension is called accreditation, which is the process of giving simulation models an official certification that shows if the simulation models' data can be regarded as acceptable for use within a specific purpose.

2.2 Existing verification and validation methodologies

Sargent [6] presents several V&V techniques that can be used to validate simulation models, together with a recommended minimum procedure for validation of a simulation model. The procedure is described in eight steps, but due to that these steps should be seen as the bare minimum according to Sargent, the procedure can be deemed too generalizable to be seen as a useful framework. However, some of the validation techniques that Sargent [6] discusses can be of interest for the goal of this thesis and these techniques are listed below:

- Animation
- Comparison to other models
- Event validity
- Face validity
- Extreme Condition Tests
- Traces

A proposed framework for optimizing V&V of simulation models has been created by Roungas et al. [12]. They present a framework under the assumptions that simulations, real-world systems, methods, and techniques are dependent on the circumstances of the study performed. Furthermore, a table of suggested V&V methods are provided, with descriptions of each method's suitability concerning source code accessibility, availability of real data, game V&V study suitability, types of requirements and purpose of the study. The framework incorporates the four validation process phases introduced by Sargent [6]. In addition, the framework contains a list of suitable statistical techniques to validate or verify various kinds of datasets. The V&V method selection methodology proposed by Roungas et al. [12] utilizes the aforementioned model properties and characteristics to determine appropriate V&V methods in regard to the validation and verification purposes.

Barceló [13] introduces validation on models as an iterative process that is used to calibrate the model parameters and compare the model to actual system behavior. The seen differences are used to gain insight on how to improve the model until an acceptable accuracy has been reached. The calibration process performed during the validation has the objective to find the values for parameters that produce a valid model. Therefore, according to Barceló, validation is concerned about if the simulation model is an accurate representation of a given system. He also discusses a basic methodological approach for validation of simulation models, assuming that the input data can be properly modeled and that the measured data that the simulation output is compared to is assumed to be error-free. He continues on to discuss how the validation process consists of collecting simulated data and comparing it to measured data, to determine if the simulated model is valid or not. If it is considered non-valid, the model needs to be revised and revalidated, thus implying that validation is an iterative process that may need to be performed several times as the model changes.

To help with the validation process, Barceló presents and discusses a four-step process, that can be used iteratively after the first step has been satisfied. Steps 2-4 can be used iterative until an acceptance accuracy of the model is met[13]. The four steps Barceló discusses are:

1. Error checking
2. Capacity calibration
3. Route choice calibration
4. Performance validation

The four steps can be explained in the following terms. Error checking is the process to eliminate as many errors as possible in the model, to ensure that the model does not create inconsistencies which could create problems that should not exist. Capacity calibration is the process of determining the most appropriate and valid parameters for core functions in the model. Route choice calibration is the process to calibrate a model's routes if routing exists in the model. The last step, performance validation, is performed on the entire model, where performance measurements such as travel time are compared to existing measures.

Barceló [14] highlights that the validation of simulation starts at the verification level. He brings up the microscopic traffic simulator called SUMO, which is a simulator similar to esmini, as an example of a computer application that verifies its models at different levels. The mentioned levels are: unit tests, acceptance tests and model tests. Unit tests are the smallest test entity to verify smaller, individual functions of a software. Each function's expected behavior and output is compared to the actual results given a certain set of input parameters. Acceptance tests compare the output of the entire simulation model to the expected results. They are more difficult to implement as the testing scenarios can become complex, but they are more suitable for verifying the overall behavior of the model. Model tests on the other hand are performed by comparing a model to other similar models.

Law [15] introduces a tutorial that demonstrates techniques for how to build valid and credible simulation models. He states that a model can be considered valid if it can be used in decision-making scenarios resembling those that would occur if experimenting with the real system was sensible and cost-effective to an equal degree. Since a simulation model is an approximation of a real system, the validity of a model can never be absolute and depends on the time and money investments during the model development. In addition, a model should be developed for a specific purpose. Moreover, the credibility of a model depends on a number of factors; including what Law phrases as the decision-maker, their understanding of the model and involvement in the project, the reputation of the model developers and effective and convincing animation. Credibility is not synonymous with validity, a model can be considered credible without being valid. For example, a model can be visually convincing but not have the needed accuracy to be considered valid.

A seven-step approach is presented in Law's tutorial [15] on how to create, implement and evaluate valid simulations:

1. Formulate the problem
2. Collect data and construct assumptions
3. Verify that the assumptions are valid
4. Program the model
5. Verify the validity of the programmed model
6. Design, conduct and analyze experiments
7. Document and present the results

Communication errors are highlighted as a major reason for invalid assumptions of simulation models, according to Law [15]. In order to mitigate these errors, a comprehensible assumptions document should be constructed to act as the main source of documentation for various stakeholders. Law [15] emphasizes that the document should primarily describe the particular issues a simulation model addresses, rather than providing an extensive description of the entire system. The assumptions document should contain enough information to act as the foundation for the developed simulation computer program. A couple of relevant points to include in the assumptions document are:

- Project goals
- Process-flow / System-layout diagram
- Subsystem descriptions and their interactions
- Simplifying assumptions
- Model limitations
- Model input data summary
- Important sources of information

In addition to the assumptions document, Law [15] pinpoints that for the assumptions document can be validated by performing a structured walk-through. Moreover, he specifies that this validation technique of the assumptions document might be one of the most important validation techniques for nonexistent systems. The structured walk-through is performed by having a meeting with experts and stakeholders, and go through each point of the document and only proceed to the next point if every meeting participant is satisfied. The assumptions document should be sent to each participant prior to the meeting to allow for early reviews and comments. The main benefit of this validation technique is to ensure that simulation analysts receive a comprehensive and correct understanding of the system to model, increasing both the validity and credibility of the simulation model.

Law [15] proposes a couple of techniques to verify and validate simulation models. One proposed technique is to validate the output of a model by comparing it to observed data from the modeled, or real, system. The simulation model would then be considered valid if the model output data is close to the output data of the real system, by some threshold. This method is sometimes referred to as results validation. Statistical tests can be used to compare the data, though the common assumptions of independent and identically distributed data are not directly applicable. This is due to the nature of most real-world and simulation systems being non-stationary and auto-correlated [15]. A similar suggested statistical method is a Turing test, where system experts examine different sets of data without knowing their origin. Another proposed technique is animations. The animations can be beneficial in displaying the invalidity of a simulation model, and they can also be used to improve its credibility. Additionally, it can also be applicable for verification purposes of a simulation computer program.

David et al. [16] discusses the V&V process on simulation model and proposes several methods and techniques that can be used during the different stages of the process. They discuss that the verifiability of a simulation model is influenced by the procedure used to develop the simulation and breaks it down to two main procedures:

- The model is definable as a set of input/output pairs in a specific range and thus the corresponding model is verified for the range considered.
- The model is defined according to a researcher/stakeholder's intention in a specified range and thus the corresponding model is verified for the range considered until it reaches the researcher/stakeholder's expectation.

David et al. [16] mentions that both of these procedures create a clearly defined parameter range to be verified. To help with the verification, they discuss how techniques such as object-oriented design can simplify testing and debugging of the model and how the use of defensive programming methodologies such as assertion tests and unit test are well suited for verifying simulation models. They also specifically mention two verification methods that according to them are the most important ones to complement the techniques mentioned, these methods are traces and structured walk-through.

Raunak and Olsen [17] investigate how validation can be performed on discrete-event simulation (DES) models, from the perspective that this type of simulation is often deemed as non-testable software. They discuss that performing validation on non-testable software does not have methods for identifying and establishing the coverage of validation in regard to simulation models. When validating a simulation model, one needs to identify all elements that need validation. They propose that this can be done by organizing them into high-level aspect groups, where each aspect contains elements types. The elements can be divided into two categories, input data and observable emerging information (OEI). OEI can be any observable or measured behavior or data during the execution of the simulation [17].

For DES models, Raunak and Olsen [17] propose that the aspects groups are: Resources, Request Characteristics and Workflow, these are defined after the three main groups of elements found in a DES model. Elements in each aspect should be validated to ensure that the entire model is valid. However, they do not specifically propose any validation techniques, rather they mention that one should use the most common techniques that fit each element, such as animation, comparison to other models and extreme condition tests which are presented by Sargent [6] and Law [15]. Raunak and Olsen's [17] main contribution is the implementation of weights to the validation process to be used to calculate a coverage rate for the validation process , where different techniques are given different weights based on the level of confidence the technique has. Each element is also given a weighted score based on the importance of validation of each element, the higher importance the higher weight. To then calculate the coverage of the entire model, three sets of information are needed.

- Elements that should be validated
- Validation techniques that can be applied for each element
- Validation techniques that were applied on each element

For each element, the coverage degree is determined by combining the weighted score of each validation technique used divided by the combined weighted score of all possible validation techniques. The coverage degree is then multiplied with the weighted score of the element and after this is done on all elements, the coverage degrees are combined and lastly divided by the total combined weight of all elements to get the coverage degree on the entire DES model.

Another framework for V&V is proposed by Wang and Lehmann [18]. The frame-

work focuses on three concepts of a simulation study: an associated role concept, a process and product concept, and a quality assurance concept. The role concept establishes an organizational structure to the model development by introducing responsibility roles according to each role's capabilities, such as sponsors, model designers, software/hardware developers and users. The process and product concept is targeted towards the intermediate products developed during the different development cycles of a simulation model. The intermediate steps here refer to the products created from each role, for example conceptual models from a model designer or executable models from a developer. In the framework, the simulation model development process consists of decision points [18]. Each decision point is associated to a progress stage of the development and a collection of V&V activities to perform in each respective stage. The intermediary V&V activities are used to evaluate the intermediate products. However, the framework description does not specify any specific suitable V&V techniques to apply, but mentions that the activities should be defined along with a requirement document.

The aforementioned frameworks or methodologies fall short of the area of this thesis, since the intention of the proposed framework in this thesis is to be more focused on simulation models based on the standards OpenDRIVE and OpenSCENARIO. Sargent's [6] proposed framework can be seen as more focused on mathematical or physics simulation models, where his recommended procedure can be hard to apply to traffic and road scenarios, where the models simulate more than just physics. Roungas et al.'s [12] proposed framework can be seen as too general for the application on esmini, where some parts could possibly be extracted and used on esmini, but the entire framework is not directly compatible. Barceló's [13] idea of an iterative process could be applied to esmini, but would require modifications to be more appropriate for the models developed in esmini. Law's [15] proposed assumptions document could be directly applicable to esmini, as it would improve the understanding of the model to develop, however it would not improve the V&V process alone. The idea of adding weights to the V&V process which Raunak and Olsen [17] propose does not in itself solve the issue of V&V, as their approach is more focused on improving existing processes to increase the trustworthiness.

2.3 Modeling and Simulation techniques

Modeling and simulation techniques are research methods that can be applied to organizational and operational systems in order to improve the V&V process of model design and development [19]. They are utilized to increase the understanding of real-world system behavior.

2.3.1 Simulation techniques

In operational management systems, there are two commonly used simulation methods, agent-based simulations and discrete-event simulations.

Agent-based simulation (ABS) is a bottom-up architecture simulation method for

simulating industrial processes and complex scientific systems [19]. This form of method aims to model individual agents (entities) and their interactions in a decentralized manner, where each agent control themselves. The simulation model inputs are therefore usually based on theory and data with respect to the behavior of the agents. The individual agents' decisions affect the outcome of the overall system. Consequently, ABS is becoming a more prominent method to model and simulate systems where behavior is heavily dependent on the behavior of smaller system entities.

On the contrary, discrete-event simulation is a top-down architecture simulation method where the focus is put on modeling time-based behavior systems on a higher level in a centralized manner [19]. The model input data is frequently based on objective data collected from the system. Each entity in the simulation is processed in sets of related steps dependently, where the simulation behavior is determined by the relationship of these steps.

2.3.2 Traffic modeling techniques

The purpose of traffic modeling is to reproduce real-life observed traffic. Traffic modeling contributes to the planning and management of traffic in road networks [20]. In general, the traffic simulation models target three main areas of traffic dilemmas: traffic flows, road network elements, and time and cost estimation of travels. Traffic simulation models can be divided into three categories of modeling architectures and applications, these are: microscopic, macroscopic and mesoscopic modeling.

Microscopic modeling is based around modeling attributes of vehicles and their movement in traffic. Various parameters such as traffic flow, vehicle speed and stops are taken into account when collecting the model data. Azlan and Rohani [20] provides brief descriptions of car following, lane changing and gap acceptance models as examples of microscopic modeling. In each of the models, certain parameters are observed to allow individual vehicles to control their own behavior and perform suitable actions.

Macroscopic modeling, models the traffic stream from mainly speed, flow, and density characteristics in mathematical descriptions [20]. These forms of models are considered continuous simulations. Macroscopic models apply the same treatment to every vehicle, and thus cannot make independent actions for each individual vehicle [21]. This form of modeling is useful for models that do not require a high level of detail, for example highway networks. The main benefit of macroscopic modeling is its potential to provide useful and accurate information while still enabling a fast simulation.

Mesoscopic modeling can be seen as a middle ground between macroscopic and microscopic modeling [22]. It provides analysis of the different entities in smaller groups [20]. The level of detail in the modeling of individual traffic entities is higher

than macroscopic, but not as high as microscopic.

There exists a fourth category, which is nanoscopic or submicroscopic modeling. An extension of the microscopic modeling approach, where details regarding internal functions of a vehicle is taken into consideration. This type of modeling is often used for autonomous driving simulation, where vehicle vision or details like gear shifts are simulated [22].

2.4 ASAM Standards

Association for Standardization of Automation and Measuring Systems (ASAM) is a non-profit organization that hosts standards within the field of automotive development and testing, founded in 1998 after an initiative by several German car manufacturers, such as AUDI and BMW [23],[24]. As of now, ASAM has more than 350 companies that are members in the organization. The standards that ASAM hosts are developed in collaboration between the members.

ASAM Standards defines data models, communication APIs, file formats, software components APIs and communication protocols for data exchanged to help research, development, and validation [25]. The aim of ASAM standardization is to allow for the choice of the best tools based on capabilities, efficiency and support.

2.4.1 OpenDRIVE

Open Dynamic Road Information for Vehicle Environment (OpenDRIVE) is one of the standards that is created by ASAM members, where the goal is to create a common base for describing road networks for driving or traffic simulators [26]. The OpenDRIVE standard version 1.7.0 defines a format for creating road networks using extensible markup language (XML) syntax and the file extension xodr. The main purpose of OpenDRIVE is to allow for exchanges between simulators, this so that the same description can be used without the need of first translating it, thus reducing cost and spent time for development in the industry.

OpenDRIVE can be used to describe geometry of roads, lanes, and objects such as road markers, it also allows for defining features such as signals. Roads in OpenDRIVE are defined based on a reference line, which has its own coordination system (s/t) which differ from the absolute coordination system (x/y). This allows for features such as road markers to be attached to a road's reference line with (s/t) coordinates. Lanes are attached on either side of the reference line, given +/- IDs based on which side of the reference line the lane is [27], an example of this can be seen in Figure 2.1.

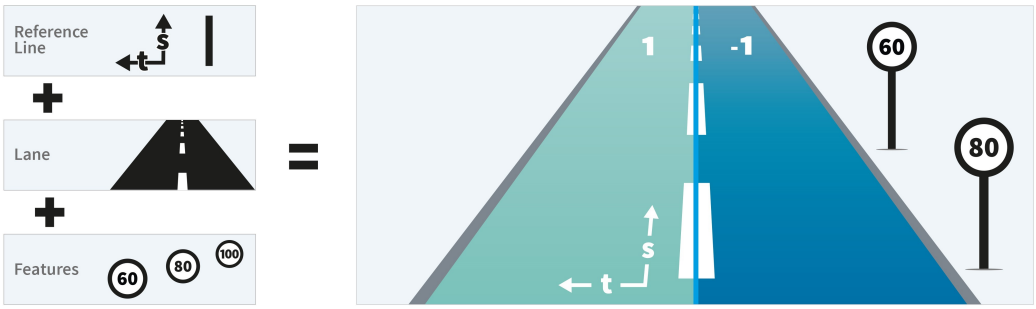


Figure 2.1: Example of OpenDRIVE reference line [26] © ASAM e.V.(Used with permission)

Roads can be connected to form junctions, and roads and junctions combined can form road networks. Junctions can be defined in three different ways according to OpenDRIVE, these are: common, direct or virtual junctions [27]. Common junctions contain several overlapping roads that are drivable, which can represent a normal four way intersection, seen in Figure 2.2. Direct junctions cannot have overlapping lanes and can be used to represent highway entries and exits, seen in Figure 2.3. Virtual junctions are not implemented in the current version of esmini, hence they are irrelevant for this thesis.

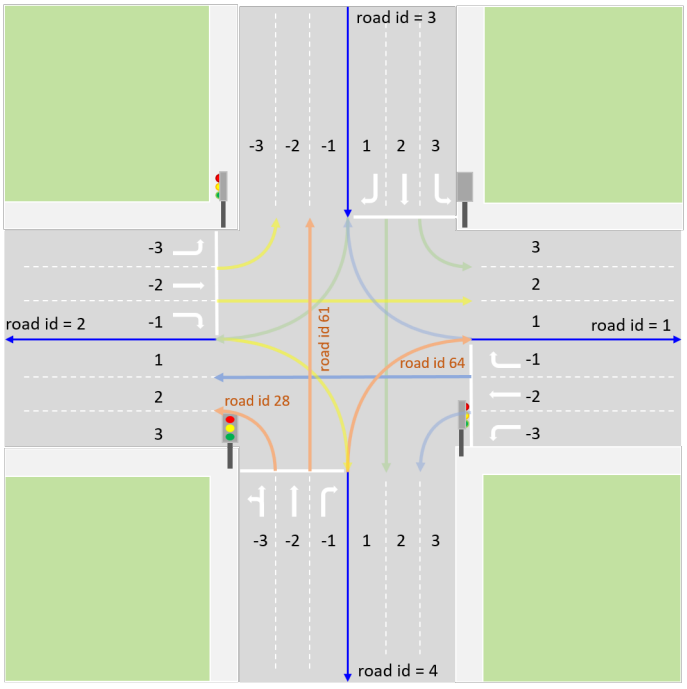


Figure 2.2: Example of OpenDRIVE common junction [27] © ASAM e.V. (Used with permission)

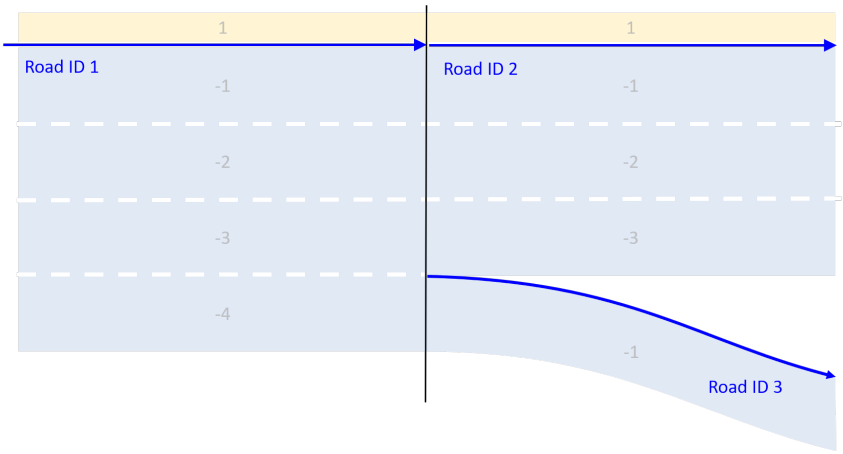


Figure 2.3: Example of OpenDRIVE direct junction [27] © ASAM e.V. (Used with permission)

2.4.2 OpenSCENARIO

OpenSCENARIO is another standard created by ASAM members, it defines a file format using XML syntax to describe dynamic content of traffic and driving scenarios. The OpenSCENARIO version referenced during this thesis is 1.1.1 [28]. OpenSCENARIO's primary use-case is to describe complex maneuvers that involve several entities, where a maneuver is described as driver actions or trajectories that are performed synchronously. However, OpenSCENARIO is not limited to this, as it also can contain the description of vehicles, drivers, pedestrians, traffic and environmental conditions [9].

A maneuver is described in a storyboard, which is divided into three parts, stories, acts, and maneuvers groups. A story describes a specific driving maneuver, for a single vehicle, or specifies the dynamic behavior of several entities. Each story can then be divided into several acts, which can be set to trigger at specific conditions. Each act contains a maneuver group, which contains maneuvers on multiple vehicles [9].

These maneuvers are described in detail by the use of events and actions, where events describe when something is supposed to happen and actions describes what should happen. There exists three types of actions in OpenSCENARIO, private, global or user-defined. A private action describes what should happen for a single entity, such as a vehicle, and could be a lane change. A global action relates to the simulation and can be used to modify non-entity elements, like time of day and traffic signals. A user-defined action however is an action that is not part of the standard OpenSCENARIO, but something added between the simulator and the scenario designer[28].

2.5 Environment Simulator Minimalistic (esmini)

Environment Simulator Minimalistic is a simulator developed during a Swedish collaborative research project called Simulation Scenarios [29], the development of which has continued based on user needs and the continued development of OpenSCENARIO and OpenDRIVE [9], [30]. It is mainly written in the C++ programming language. Currently, Volvo Cars is one of the main developers of esmini. The two most important modules used in esmini are RoadManager and ScenarioEngine. RoadManager provides the interface to OpenDRIVE, while ScenarioEngine provides an interface to OpenSCENARIO. The purpose of esmini is to provide a cross-platform development tool for working with the OpenSCENARIO data format [30]. Traffic scenarios in esmini are simulated from road network information stored in OpenDRIVE and OpenSCENARIO formats. The modeling techniques DES and microscopic modeling are used in esmini, and these concepts are explained in Section 2.3.

Since esmini a deterministic simulator with little stochastic models, it can be used as a foundation for building other simulators that implement more stochastic features as it can act as the "ground truth" for elements of the simulation that are not as important for the objective of the simulation. The esmini simulator is therefore useful when only the barebone of a simulation is necessary.

2.5.1 Structure

There are six main modules in esmini, these are RoadManager, ScenarioEngine, Controllers, ViewerBase, PlayerBase and CommonMini [31]. As mentioned previously, the RoadManager module is responsible for the implementation of the OpenDRIVE format. ScenarioEngine handles parsing of OpenSCENARIO files and constructs entities, triggers and actions which are then consequently executed. The ScenarioEngine also accompanies a ScenarioGateway, whose purpose is to asynchronously provide information about entity states, for instance through the UDP protocol. Controllers are a concept present in OpenSCENARIO that enables custom behavior of individual entities in addition to the functionality provided by the default controller within ScenarioEngine. Controllers follow an API defined by class inheritance, where inherited classes can override and define custom behavior for certain entities in the scenario. The ViewerBase module is a 3D viewer that provides visualizations for the road network and features, as well as 3D models for the scenario entities. The PlayerBase module connects the ScenarioEngine and ViewerBase to provide a higher level API for handling scenarios in custom player applications. Lastly, the CommonMini module is a collection of useful functions that can be used by other modules. Some examples of the functionality provided in collection are timers, math operations and loggers [31].

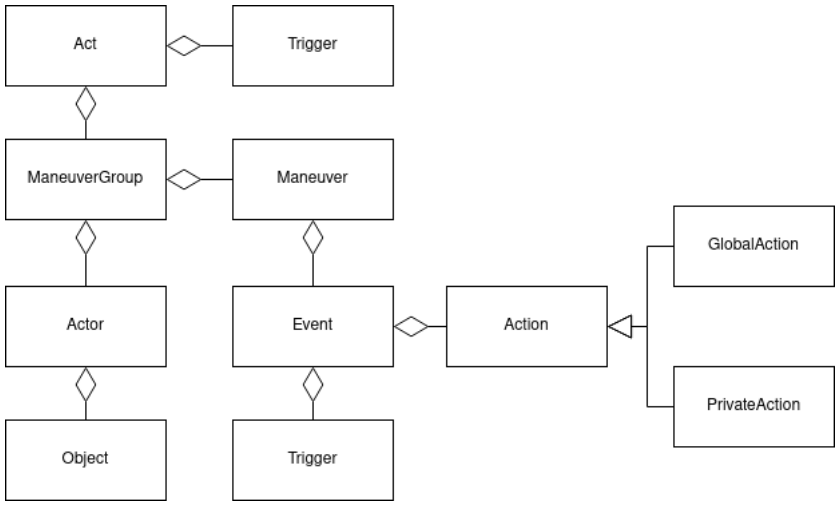


Figure 2.4: UML diagram of acts and all related classes in esmini.

2.5.2 Event and Action

Events and actions in esmini are implemented based on the definitions from the OpenSCENARIO standard [28]. As previously mentioned, OpenSCENARIO defines three different actions, however only two of them are implemented in esmini as of spring 2022. The two actions that are implemented can be seen in Figure 2.4, which shows how all the classes related to acts are linked in esmini. In the UML diagram, it can be seen that events can have one or more actions and a trigger that states the start conditions of the event. It can also be seen that all links between these classes are unidirectional, meaning that e.g, actions can not see what events they are a part of.
Global and private actions work on different parts of the simulation, global on the simulation and private on an entity. The implemented global actions are:

- Set parameter
- Traffic swarm
- Environmental / infrastructure
- Add entity

The private actions implemented in esmini are:

- Longitudinal speed
- Longitudinal distance
- Lateral lane change
- Lateral lane offset
- Assign controller
- Activate controller
- Teleport entity
- Assign route
- Follow trajectory
- Synchronize

Most of the private actions in esmini are quite self-explanatory of the functionality

based on the name of the action. The actions have different parameters that change their behavior, one of these parameters are if the actions has a relative or absolute target. E.g, in lateral lane changes, this changes the behavior to either change to a relative lane based on the location of the entity or an absolute lane that the entity should change to.

2.5.3 Controllers

Controllers, which is a core element of the OpenSCENARIO simulation [28], is implemented in esmini. The purpose of a controller is to customize the behavior of simulation agents during runtime. A motivating factor for the usage of controllers is that they provide a flexible way of extending functionality in esmini while also conforming to the OpenSCENARIO standard [32]. The standard does not define controller implementations, and the implementation present in esmini is hence an interpretation of the controller concept. However, the standard does specify that controllers are deactivated by default. Furthermore, esmini provides a set of embedded controllers that can be useful for simulating certain scenarios. For instance, the SloppyDriverController is a simulation model where the simulation entity performs slight speed and turning deviations in order to mimic a sloppy driver [32].

Controllers are activated for a domain: longitudinal, lateral or both [32]. A domain represents the direction of motion of a simulation entity. The longitudinal domain corresponds to motion along a reference line, for example, increasing the speed of the vehicle. The lateral domain corresponds to motion perpendicular to a reference line, such as lane changes.

OpenSCENARIO specifies two controller types: the default controller and user-defined controllers. The default controller must enforce so-called control strategies precisely, while user-defined controllers may customize these strategies. Control strategies define how entities are assigned behaviors [28]. The default controller follow control strategies after how they are defined in the OpenSCENARIO file.

2.5.4 Similar simulators

There are two other traffic simulators that are similar to esmini in the terms of being capable of using the ASAM standard OpenDRIVE. These two simulators are CARLA [33] and SUMO [34]. CARLA is a open-source simulator developed for autonomous driving development and uses high fidelity graphics and physics enabled by the use of Unreal Engine 4 to ensure that it is possible to train sensors with realistic data. This means that CARLA is quite heavy to run compared to esmini. CARLA have some support for OpenDRIVE and OpenSCENARIO, but does not currently support the most recent version of OpenSCENARIO. SUMO is a more lightweight traffic simulator and focuses more on traffic flows and traffic management, which differs from esmini that is more useful for simulating smaller traffic cases. The main difference between esmini and the simulators CARLA and SUMO is that esmini was purposefully developed to support the ASAM standards

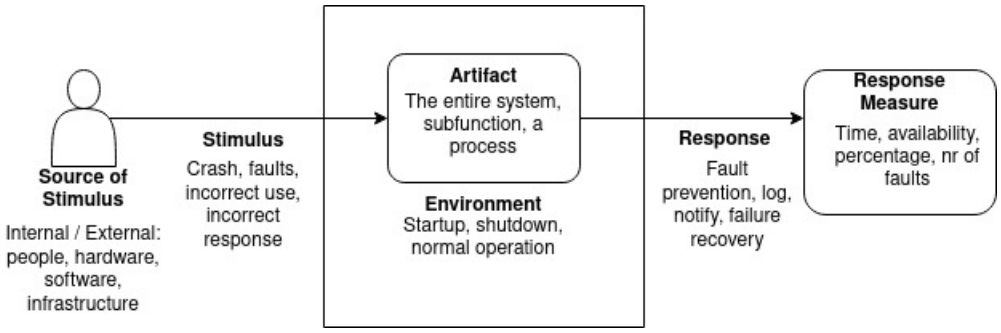


Figure 2.5: Example of a general QAS for the quality attribute availability.

OpenDRIVE and OpenSCENARIO, while the other simulators has support for these standards by the use of additional components but it is not one of the main features.

2.6 Quality Attribute Scenarios

Quality attributes are used in software engineering to describe a measurable or testable property of a system [35]. The ISO 25010:2011 Standard [36] defines a set of software quality attributes to use together with requirements and quality evaluations. The attributes defined in the standard are the ones referred to in this thesis. A quality attribute is used to indicate how well a system reaches its requirements from stakeholders [35]. These are often retrieved from descriptions such as functional requirements of a system, which means that they often specify attributes, like modifiability or availability, that can be hard to measure without further explanation or requirements. To solve this, one can use quality attribute scenarios (QAS), which contains more concrete information about what is expected of a system. A QAS is divided into six parts, each describing one section of the requirement [35]:

- Stimulus: An event that arrives to the software.
- Stimulus source: Where the stimulus came from.
- Response: How the system should respond to the stimulus.
- Response measure: Measure of the response, if it reached the required degree.
- Environment: The condition of a system when the stimulus happens.
- Artifact: The portion of the system that is applicable for the requirement.

QAS allows for making quality attribute requirements testable and can be divided into two subgroups, general or concrete, depending on how the QAS is defined.
General QAS are quality attributes scenarios that are not bound to a specific system [35]. These are defined with a collection of definitions for each part of the QAS which contains the characteristics for a specific quality attribute. A general QAS can be modified to be system dependent by just having the relevant definitions for a quality attribute for the given system. An example of a general QAS for the quality attribute availability can be seen in Figure 2.5.
Concrete QAS (CQAS) are however specific to a system [35]. These are used when testing if a system reached the wanted degree of a certain quality attribute. The

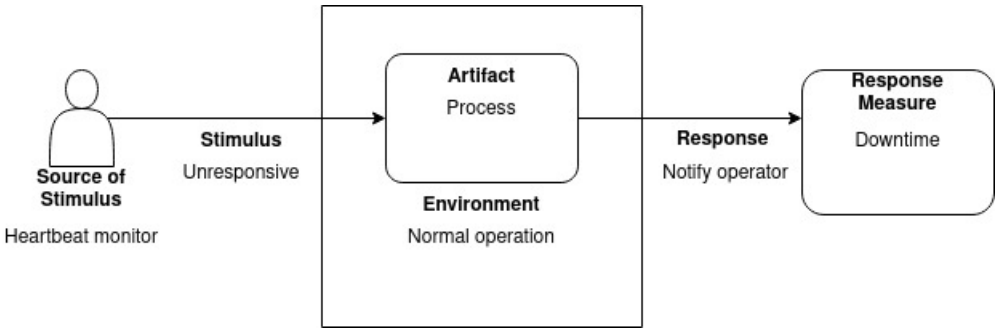


Figure 2.6: Example of a concrete QAS for the quality attribute availability.

difference between a general and a concrete QAS is that the CQAS only have one definition for each part, which allows it to create a testable definition of how a system should react to an event, as well as making it measurable. An example of a CQAS for the quality attribute availability can be seen in Figure 2.6.

2.7 Acceptance Test-Driven Development

Acceptance Test-Driven Development (ATDD) is a development methodology that is similar to regular test-driven development, but focuses on acceptance testing to help improve software quality and ensure that customer needs are fulfilled [37]. ATDD helps to provide a software development structure that closely follows and implements the required functionality. Furthermore, the productivity of a team and the quality of the code are increased by applying ATDD in a team [38]. The main difference between a typical development workflow and an ATDD workflow can be seen in Figure 2.7 and Figure 2.8. This section aims to describe the concept of ATDD and how it can be beneficial in a V&V context.

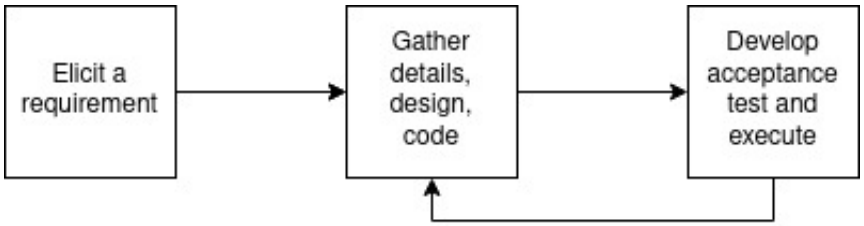


Figure 2.7: Flowchart of a typical development workflow

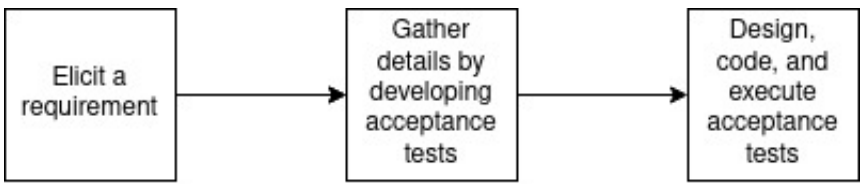


Figure 2.8: Flowchart of acceptance test-driven development workflow

The main building blocks of ATDD processes are user stories [37]. Briefly described,

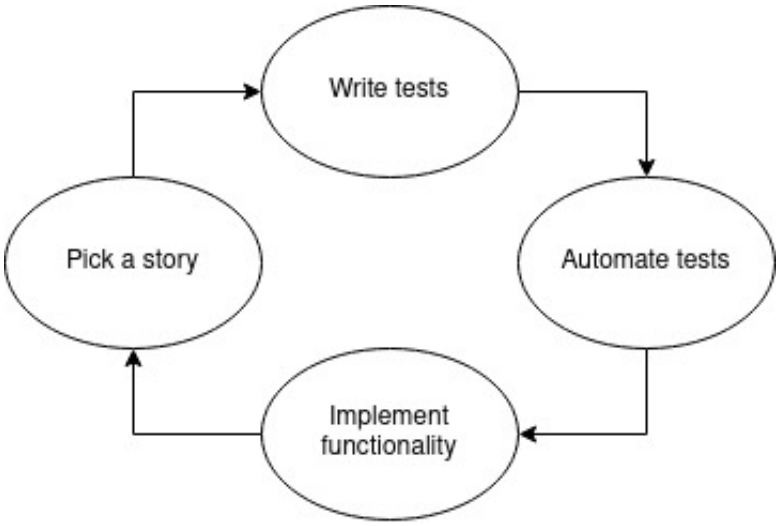


Figure 2.9: The ATDD cycle visualized as a graph

a user story is a format for describing requirements that expresses desired behavior by specifying who does what and why. In other words, the who can be seen as a role or entity, the what can represent some functionality and the why describes the benefit of the functionality. User stories are convenient for specifying requirements in a format that most project participants can comprehend.

Acceptance tests are high-level descriptions of some desired system behavior or functionality [37]. These tests are designed in relation to a user story and establishes the behavior of the system in regard to some set of conditions as well as input and output data. Acceptance tests focus mainly on what functionality to implement and not on how it is achieved. An important aspect of acceptance tests is that they are usually written together with different project roles, where the customer, developer, and tester are all contributing to the design of the test.

The ATDD process is performed cyclically, and the cycle consists of four steps. A basic visual representation of the cycle can be seen in Figure 2.9. A more detailed visualization can be seen in Figure 2.10.

1. Pick a story
2. Write tests
3. Automate tests
4. Implement functionality

This process is performed for every user story. First, a user story is picked, and the choice of story can depend on a priority list of user stories to consider. Secondly, create the acceptance tests together with the customer, for example by spontaneously coming up with relevant scenarios to test and sketching them down in any shape or form. Thirdly, the tests should be automated so that they can be executed automatically, making sure that the tests are identically executed every time and that they provide simple yes-or-no answers. The fourth and final step in the cycle is to implement the functionality in order to pass the previously created tests. When all

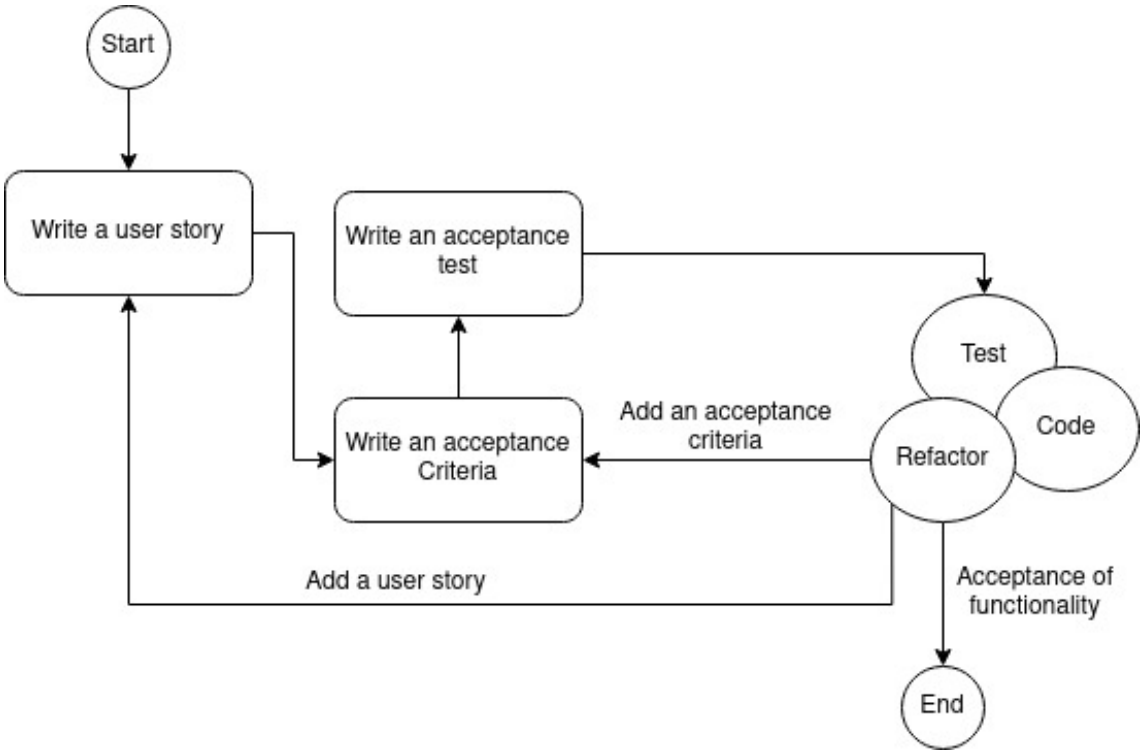


Figure 2.10: The ATDD cycle graph in a more detailed visualization

the tests are passed, by implementing the corresponding functionality, a new user story is picked and the cycle is repeated accordingly [37].

One of the main benefits of an ATDD workflow is the distinct definition of done it provides. Koskela [37] highlights two key issues of project management, knowing where a team is in a project timeline and knowing when to stop. ATDD provides a simple answer to these questions, which is that a requirement is fulfilled if all the tests for the corresponding user story pass. If not, there is remaining functionality to be implemented before proceeding to a new user story.

2.8 Test automation

Test automation is usually associated with the automation of test execution, but has in more recent years been expanded to other test activities such as the design, scripting, evaluation and reporting of tests [39]. Several tools and software libraries already exist to provide effective help for automation in different testing activities. The automation of testing activities aim to reduce the effort and cost required to develop tests, which subsequently promotes development of higher quality software.

An example of a testing approach to automate both test design and generation is model-based testing (MBT) [40]. Instead of letting a developer manually design a test, MBT uses modeling languages such as UML to express the behavior of a system. Various model elements, such as states and transitions, are traced from the

requirements to describe the system flow and its decisions. For most MBT tools, tests can then be automatically generated from the model, with input parameters and expected output parameters. The generated test describe sequences of actions and events that a developer would normally have to perform manually. The essential modeling process of MBT creates a visual representation of the model, usually in the form of a graph. This promotes understanding of the model by not only developers, but also testers and other stakeholders. Furthermore, the MBT test generation criteria can be altered to limit the number of generated tests [40]. For instance, they can focus on requirements, allowing more tests that prioritize requirements to be generated to verify that all requirements are fulfilled. This encourages close development with specified requirements, which ensures a more reliable V&V process and ultimately higher quality software.

Another example of a testing approach to automate test generation is property-based testing (PBT) [41]. PBT generates random tests through description of valid inputs and expected properties to hold for all valid inputs. Therefore, testers only have to specify valid inputs and properties instead of manually verifying inputs and expected outputs for many test cases. Writing tests with PBT is fast, more concise, and generates more test cases than a human would be able to do manually [41].

Gambi et al. [42] propose a search-based testing technique, together with procedural generation, to automatically test self-driving cars. In summary, their idea is to create challenging virtual scenarios to test a virtual vehicle in, by generating new road networks procedurally. Challenges introduced in a scenario could be factors such as weather conditions or complex road shapes. In Gambi et al.'s case, this testing technique is used to ensure self-driving cars perform lane keeping correctly in a virtual environment.

3
Methods

This chapter will describe and motivate the choices made during the development of the framework and the model. It will also present the research and workflow techniques used to achieve the result.

3.1 Design Science Research Process

The design method chosen for this thesis is the Design Science Research Process (DSRP), as it has several advantages for our study. Mainly, DSRP allows us to develop artifacts (a framework for V&V). DSRP is a six stage process and uses an iterative process, seen in Figure 3.1, where the fifth and sixth step checks the current system and allows for improvements if deemed necessary [43][44]. DSRP will also allow us to solve a real problem, which in our case is to improve the V&V process for development in esmini.

First stage: The first stage is problem identification and motivation. The problem to be investigated in this thesis is to improve the V&V process for development in esmini using the ASAM standards OpenDRIVE and OpenSCENARIO. The research methods applied here are field study and desk research. These methods will be applied in order to gain knowledge on existing research as well as the systems involved. In this stage, the modeling techniques used for lane changing behavior in esmini will be identified. The identified techniques will create the foundation for a new artifact that is to improve V&V process. Model-based performance analysis can be leveraged to explore performance properties of esmini models. Moreover, this analysis would help us understand the modeling of parametric dependencies for better performance prediction [45].

Second stage: The second stage is the definition of the objectives of a solution. The objective is to create a framework artifact for V&V on simulation models for the esmini simulator. Other similar proposed frameworks will be examined to understand the potential steps that are feasible in the framework artifact. The artifact should primarily focus on lane changing models, but the desire is for the framework to be generalizable.

Third stage: The third stage is design and development. During this step, the creation of the artifact (a framework in the form of an informal process containing a set of systematic procedures applied in given steps) takes place (RQ1). Defining

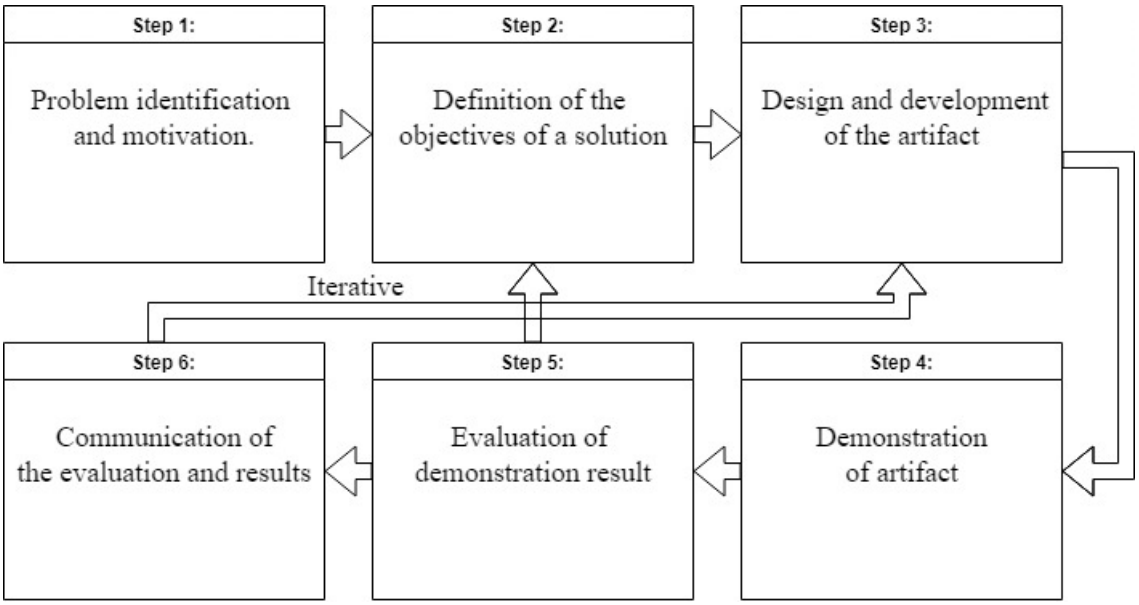


Figure 3.1: The six steps of the Design Science Research Process.

desired properties, planning of the artifact's architecture and the creation will take place during this step. The artifact will be designed based on information gathered from esmini as well as literature of existing similar frameworks. During this stage, we will get support from Volvo Cars in the form of details of simulation models in use and documentation on algorithms and access to experts in the field (esmini). Therefore, all of these resources combined should help us define the details of our framework development.

Fourth stage: The fourth stage is demonstration. In this step, the artifact will be deployed on a simulation model (RQ2). The artifact will be used to drive the V&V process on the specific model, and relevant data will be collected regarding the artifact's usefulness and effectiveness.

Fifth stage: The fifth stage is evaluation. In this stage, data collected during the previous stage, the demonstration of the artifact (the framework), will be used to determine the artifact's usefulness and effectiveness. Meaning, how much help the framework provides during the process in terms of ensuring that the stated requirements and users needs are met. At the end of this stage there will be a decision determining if the artifact has reached an acceptable level, if it is deemed to not have reached the sought after efficiency, the processes can revert to the second stage for additional adjustments. However, if deemed acceptable, the process can be moved to the sixth and last step. (RQ2)

Sixth stage: The sixth and last stage is communication. During this stage, the artifact together with the problem is communicated to relevant audiences, as for example researchers, professionals, and practitioners. In our case, the outcome will be communicated through a thesis that will present the framework and how it improves V&V.

3.2 Research methods

The research methods chosen for this thesis are field study and desk research. Field study refers to research conducted in a specific, real world setting to study a specific software engineering phenomenon [46], which for this thesis are V&V of a specific simulation software (esmini). This research method was chosen as it would allow for information gathering about the existing system in regard to modeling techniques and V&V techniques. A base knowledge needed to be established for the V&V Framework development. Therefore, the plan was to investigate the modeling techniques that were used in the simulator esmini, especially with regard to Volvo Cars' lane changing road scenarios. Desk research is a secondary study and aims to summarize or synthesize existing research presented in primary studies [46]. The reason for choosing desk research as one of the methods is that to gather knowledge for RQ1 & RQ2, we need to study and summarize already existing frameworks in regard to the V&V process for simulation models.

3.3 The creation of the framework

In this section, the process of creating the framework will be described, as well as the motivation behind the choices made during the development. It will also present the metrics used to determine the frameworks usefulness and effectiveness.

3.3.1 Development process

In order to create the framework, a structured process that could help guide the direction of development was needed. This chosen structure was DSRP, as described previously in Section 3.1. As described in the DSRP process, the first step for developing the framework was to identify the problems of esmini and V&V. To identify the problems, more knowledge of existing V&V techniques and methods was needed, so the first step was to gather more knowledge of V&V and the structure of esmini. After the knowledge was acquired, the next step was to begin to define the definition of a solution to the V&V process in esmini. During this step, the most relevant techniques and ideas were extracted and combined with the ideas of QAS and ATDD to form the beginning of a solution for V&V in esmini.

After the definition of a solution is defined, the next step was the design and development iterations, step three to six in Figure 3.1. The first iteration began with combining the ideas of Barceló [13],[14], Law [15], QAS [35] and ATDD [37], [38] to create the first draft of the framework. After the first draft was made, the next step was to use in a concrete case, to make it more understandable and to see what was missing in terms of verification and validation. The concrete case used for this is the lane independent routing model, since the framework creation is further than the development of the model, some data, and ideas used in the test application of the first draft were fictional. This process repeated itself over a number of times before the idea of the framework was finalized. The application of the framework, on the concrete case, can be seen in Chapter 5.

3.3.2 Motivation for chosen procedures

The assumptions document, based on Law's [15] idea, is important in relation to esmini mainly because of how esmini is developed with regard to the OpenDRIVE and OpenSCENARIO standards. The standards do not always specify exactly how some functionality is implemented, therefore requiring interpretations of how it could be achieved specifically in esmini. Along with the interpretations, assumptions will be needed to create a compliant implementation. Due to how esmini is implemented, there are some limitations to what can be done and with the help of assumptions the limitations in esmini can be circumvented. Additionally, the idea of using an assumptions document, or requirement document, as part of the V&V process is backed up by Wang and Lehmann [18].

Requirements are vital to the testing process of all software, as they document the desired functionality. To verify the stated requirements, different testing techniques can be used, one being acceptance tests. The creation of acceptance tests has an increased difficulty compared to other testing levels, as it can be hard to break down the requirements into testable components. The use of CQAS allows for a clearer connection to acceptance tests, where the focus of CQAS is on creating requirements based on quality attributes such as correctness or availability. Consequently, the acceptance tests are easier to define as they target a specific attribute. Additionally, CQAS reduces the complexity of behavior and functional testing by providing an example that makes the behavior or functionality description easier to comprehend. The use of requirements is also supported by David et al.'s [16] idea that models are verifiable after the stakeholder's intentions, and should therefore be verified until it reaches the stakeholder's expectations.

ATDD is a development structure that helps to closely follow and implement the desired functionality. The idea of implementing the ATDD methodology into the framework is motivated by the fact that validation becomes more prioritized and a part of the development process. Since esmini does not currently have a designated validation process, ATDD creates a structure to build the validation process upon. Acceptance tests provide additional definitions of done, allowing a team to know where they are on the project timeline and knowing when a requirement is fulfilled.

The combination of requirements, CQAS and acceptance tests through ATDD enables a substantial and iterative V&V process, where new requirements can be added throughout the development of the model. This promotes the robustness, reliability, and correctness of the simulation model. Moreover, each stage of the iterative process can be seen as what Wang and Lehmann [18] define as a decision point. Specific V&V techniques are applied during each stage in order to evaluate the intermediary artifacts created during the model development. In addition, the different development stages follow the role concept introduced by Wang and Lehmann [18], where each stage is connected to a set of roles that are responsible for that part of the model development process.

The idea of performing V&V iteratively comes from Barceló [13], as he states

that validation can be performed iteratively until an acceptable accuracy has been achieved. The iterative process allows for simulation models to be calibrated and optimized based on the requirements and acceptance criteria that has been created in earlier stages of the process. This is useful for esmini, where accurate models are especially important due to esmini being used as a test tool for safety critical software for vehicles.

The final validation step in the iterative process relates to the model tests mentioned by Barceló [14], where models are compared to each other or expected data. Here, methods such as statistical comparison, empirical analysis, and visual inspection can be used to calculate the model's accuracy. Other methods for statistical and visual validation can be obtained from Roungas et al.'s paper [12]. The addition of the final validation step enhances the validation process by including other ways of validating requirements and acceptance criteria. Using a diverse set of validation methods increases the probability of developing a highly accurate model. If the model fulfills the acceptance criteria it can be seen as validated, if not, revert to the assumption document to find why the model does not behave as expected.

3.3.3 The application of the framework

The framework will be applied to the simulation model developed as a part of the thesis, the application of the framework on the model will be shown in Chapter 5. The application will then be evaluated based on the metrics: usefulness and effectiveness to answer RQ2.

The evaluation of the framework will be based upon the example of how the framework can be used, found in Chapter 5. The evaluation will consist of two main methods, where one is the injection of faults into the model to see how effective the framework is on detecting the faults. The other is user interviews, where developers of esmini will be asked to answer a series of questions to determine if they find the framework useful or effective for esmini based on the example provided. Based on the result of the injection of faults and the feedback from developers, a degree of usefulness and effectiveness should be possible to determine. However, a study, where the framework is used during development of models by esmini developers and then evaluated based on their feedback, would have provided a stronger foundation for evaluation. But due to time and resource limitation, it was deemed not possible for the thesis.

3.4 The development of a lane independent routing model (LIRM)

This section will present the methods and steps followed in order to develop the lane independent routing model (LIRM), based on the case provided by Volvo Cars, seen in Appendix A.

3.4.1 Model criteria

The simulation model developed simultaneously as the framework is a simplified model of a vehicle navigator that should be able to find and calculate a path to a target anywhere on the simulated road network. A detailed case description can be seen in Appendix A. The current implementation of path finding in esmini is lane dependent, meaning that the path finding can not find paths that are not directly connected to the starting position's lane, which is how the default control strategy is defined in OpenSCENARIO.
The developed model, the LIRM, will be in the form of a controller capable of overriding the default behavior of path finding in esmini. The model will be able to find paths independent of the starting lane. The model should also implement a functionality for changing lanes when it is necessary, to be able to follow the path.

3.4.2 Model design

The development of LIRM started with research into existing path finding algorithms that could be adopted to fit the model's purpose. A few alternatives were considered, such as Dijkstra's algorithm and A* search algorithm [47]. The choice fell on Dijkstra's as the path finding algorithm for the implemented model, due to the lesser complexity and easier implementation.

After the choice of algorithm was made, the work began to find out and write down all assumptions made, this included the assumption around the algorithm as well as around the implementation of the entire model, such as the simulated road network in OpenDRIVE. These assumptions can be seen in the example case for the framework, in Section 5.1. This was done both as a part of V&V process of the model, but also for documentation on why some functionality of the implementation works as it does.

3.4.3 Model development

The implementation of LIRM into esmini started as a new controller that will contain both the modeled path finding and the logic for modeled lane changes. The code for the model, LIRM, can be found on GitHub[1]. The implementation can be found in files ControllerFollowRoute and LaneIndependentRouter. The newly added controller enables scenarios to use it dynamically, adding to the behavior of the default controller and provide the intended pathfinding and waypoint following functionality.

Lane independent path finding algorithm

The new functionality of lane independent path finding needed an algorithm capable of finding the path through an OpenDRIVE road network. As mentioned, the choice fell on an implementation of Dijkstra's algorithm which needed some modifications to work for the given purpose. To implement Dijkstra's algorithm, there was a

[1] https://github.com/esmini/esmini/tree/follow_routes

need for defining what were nodes and edges in the OpenDRIVE network. Our implementation was that roads became edges and the connections between roads became the nodes. This meant that the algorithm would find all connections between two roads the vehicle needs to pass through to reach the goal. To make this approach lane independent, the implementation added a node for the connection of each lane in the driving direction on a road. The lane independent path finding supported three different routing strategies, that are also specified in OpenSCENARIO [28], and these were: shortest, fastest and least number of intersections (junctions). A simplified description in pseudocode of the implemented path finding algorithm can be seen in Algorithm 1.

```
priorityqueue pq;
list visited;
Create startNode;
Add startNode to pq;
while pq not empty do
    Get first node (n) in pq;
    if n is in visited then
        Continue;
    end
    Add n to visited;
    if n == target and target is in driving direction then
        create targetNode;
        return backTraceTargetToStart(targetNode);
    end
    for each road in connection n do
        for each lane in driving direction on road do
            Get connecting road for lane;
            Create newNode;
            Calculate weight for newNode;
            Add newNode to pq;
        end
    end
end
return empty list;
```

Algorithm 1: The lane independent path finding algorithm in pseudocode.

Lane change controller

A lane change controller was developed to allow the modeled vehicle to follow routes that included lane changes to reach the target, the logic of the controller can be seen in the flowchart in Figure 3.2. This was needed as the default controller in esmini does not have any functionality of changing lanes. However, since the normal driving and route following could be handled by the default controller, it was decided to let it control the vehicle until a lane change was necessary. This design choice was made as creating a controller that handled all driving on its own would require duplicated

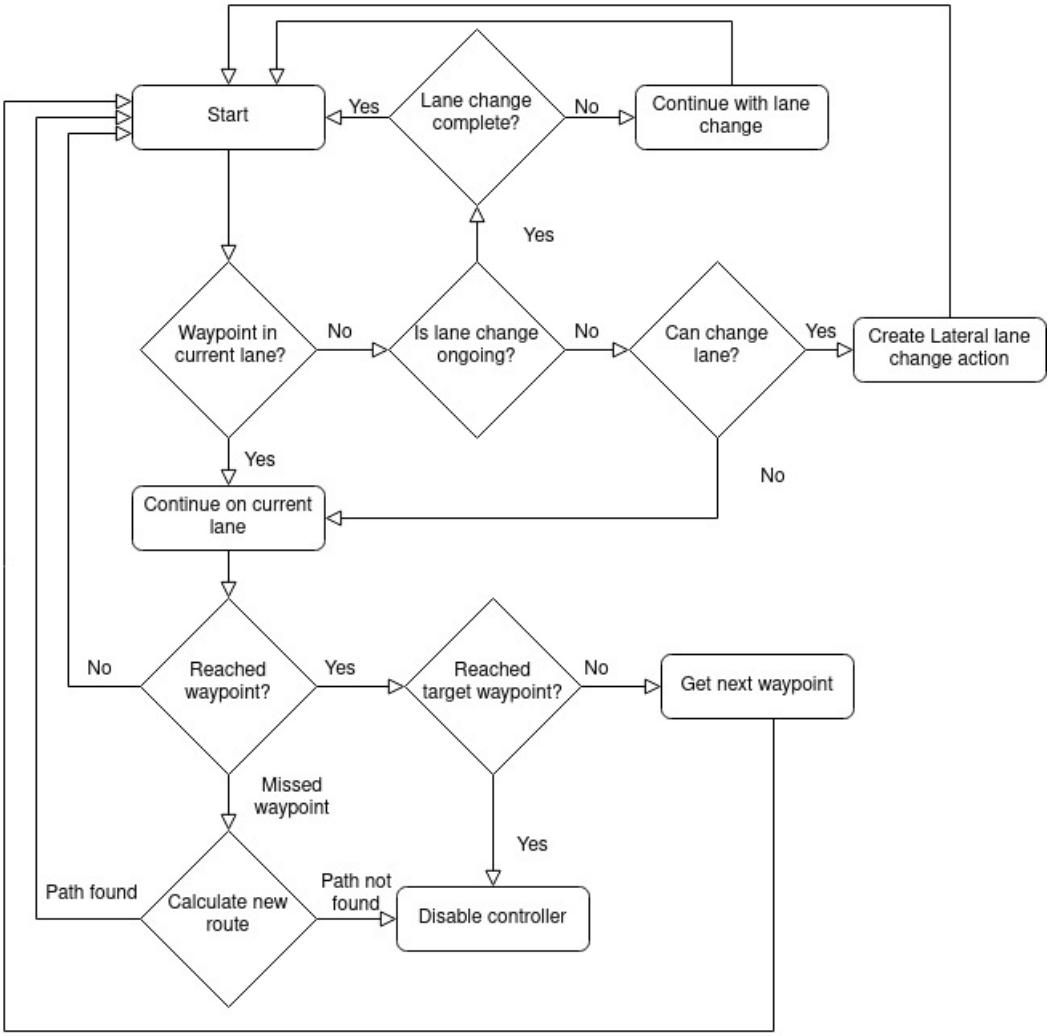


Figure 3.2: A flowchart of the logic in the lane change controller.

code and functionality that the default controller already had, which would have reduced the maintainability of esmini and should be avoided if possible.

When a lane change is necessary, the controller creates a new lateral lane change action, which overrides the default controller and takes control over the modeled vehicle while the lane change is performed. After the lane change is complete, the control is returned to the default controller.

To get the default controller to follow the waypoints created by the lane independent pathfinder, it was necessary to override the default waypoints created by the default pathfinder. This was done by continuously updating the list of waypoints that the default controller uses after each waypoint has been passed. When the final waypoint has been passed, the controller is deactivated and the default controller behavior takes over the driving of the vehicle.

4

Framework

This chapter will present the developed framework. An illustrative example of how the framework can be used to validate a model in esmini is seen in Chapter 5.

The developed framework for verification and validation for models in esmini is described in the section below. It will provide a description of how the methods and process presented in the framework can be applied to a model to improve the verification and validation. Displayed in Figure 4.1 is a flowchart of the framework. The flowchart visualizes the connections between the different steps of the framework and shows how the framework deploys the ATDD and CQAS methodologies through an iterative process.

In the current framework, all steps and processes that are a part of the framework need to be performed manually except for unit tests and some acceptance tests that can be automatically executed if wanted. The primary reason for including mostly manual steps in the framework is the time constraint of the thesis, as automating certain test activities would still require manual devising and preparation, which would have been too time-consuming.

4.1 Assumptions Document

Communication errors across stakeholders are highlighted as a major reason for invalid assumptions of simulation models [15]. In order to mitigate these errors, a comprehensible assumptions document should be constructed to act as the main source of documentation for various stakeholders. The document should primarily describe the particular issues a simulation model addresses, rather than providing an extensive description of the entire system. The assumptions document should contain enough information to act as the foundation for the developed simulation computer program. The document can be validated by performing a structured walk-through, where a meeting is held with stakeholders to ensure they are all satisfied with the stated information and that they understand the fundamentals of the modeled system. The term model in the assumption document refers to the simulation model that should be developed, in other words, the software or code that the model consists of.

The assumptions document can be seen as quite similar to a requirement document, that is often used in software engineering, as both explains the wanted functionality

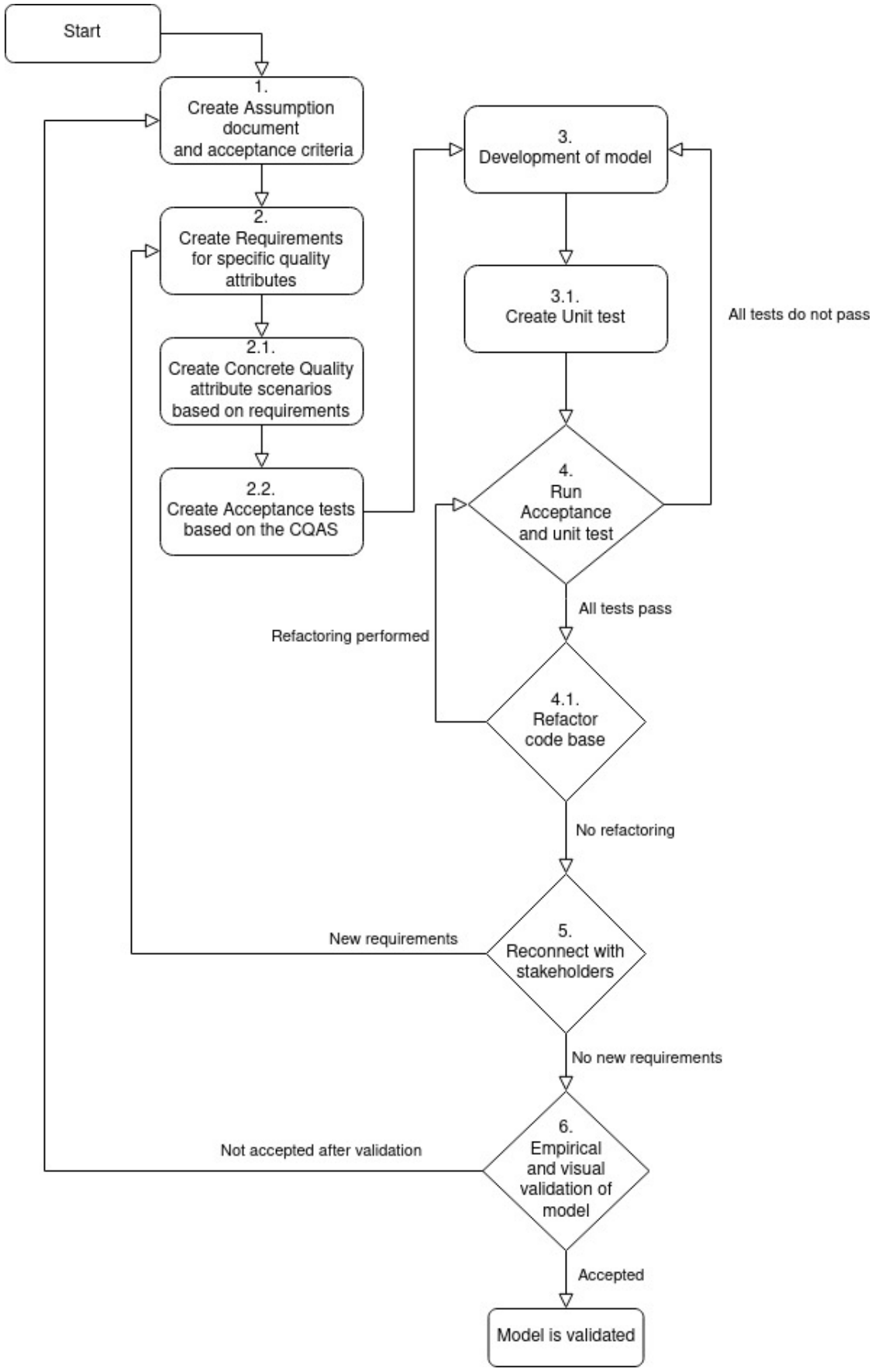


Figure 4.1: Flowchart of the V&V framework

of the software that is to be developed. However, a requirement document represents the functionality that a system should have by stating requirements that the developed system should fulfill. The definition of an assumptions document presented in this thesis aims to represent both the general functionality of a model and all possible assumptions and constraints which could affect the model development. The assumptions document should then be used during the creation of the actual requirements to ensure that the stated requirements consider the assumptions and constraints that could affect the adaptation of the real system to a model.

4.1.1 Model functionality

Provide a description of the functionality that is desired with the model usage. For example, write down a list of features, explain the model's behavior output depending on inputs, or show a figure that contributes to the overall comprehension of the model's behavior and purpose. The functionality described here does not need to be excessively formal, as the more formal descriptions are created during a later requirement stage of the framework application.

4.1.2 Model flowchart

Create a flowchart of the model's functionality. It should contain the most important functionality for the model and depict the different paths within the functionality. This should be done to improve the understanding of the model both for stakeholders and the model designers, to minimize the risk of misunderstandings of the model's functionality.

4.1.3 Model subsystems

Provide a more detailed description of the model's main functionalities. Define what each subsystem is responsible for and how they interact with each other, in terms of what type of data may be exchanged between them.

4.1.4 Simplifying assumptions

Here, all assumptions regarding simplification of the real system should be stated. Every assumption on how the simulation model simplifies the real system should be explained in detail of how it differs and why the assumption is needed. The documentation of assumptions is necessary for the creation of requirements and validation, as it provides important information about how the developed model is expected to behave in relation to reality.

4.1.5 Model limitations

Describe the possible limitations of the model that is to be developed. Any underlying limitations in algorithms, software, or data used should be considered and noted down. The limitations are essential in order to understand what cannot be achieved with the model or how to possibly circumvent certain constraints.

4.1.6 Model input data summary

Summarize the model input data to create a collection of data that can more easily be comprehended by different types of stakeholders. The summary acts as an overview of what input data the model is provided. It should also include the different types of data, for instance text, numbers, boolean values, files etc. Complex data should be explained and simplified to increase the level of comprehension and make it easier for stakeholders with less specific knowledge to gain an understanding of the data.

4.1.7 Important sources of information

Document the sources used to gather the information that is being used to establish the assumptions and limitations of the model. For each source, provide information of what type of information the source provides. This should be done to facilitate development in future iterations of the model, as it will eliminate the need to find where more detailed information regarding the model can be found.

4.1.8 Acceptance criteria

Acceptance criteria are used to define the definition of done for the model. The criteria should define how the model will behave related to the real system, and to what degree the model can differ while still being acceptable. These criteria are used during the V&V process to know if the model has reached the wanted degree of accuracy to be deemed validated. These are also important for the requirement and acceptance test creation for the model, in order to reconnect to the criteria and check if the model is considered verified and validated.

4.1.9 Assumption validation

Validate the assumptions document before proceeding to the next step. A suggested validation technique to use here is a structured walk-through [15]. A structured walk-through is performed by having a meeting with experts and stakeholders, and together address each point of the document and only proceed to the next point if every participant in the meeting is satisfied. If any point is deemed inappropriate, that part of the assumptions document should be reconsidered and improved accordingly. The assumptions document should be sent to each participant prior to the meeting to allow for early reviews and comments. The main benefit of this validation technique is to ensure that simulation analysts receive a comprehensive and correct understanding of the system to model, increasing both the validity and credibility of the simulation model.

4.2 Requirements, Concrete QAS and Acceptance Tests

After the assumptions document has been created and validated, the next step in the framework is to create the requirements, CQAS and acceptance tests for the

model.

4.2.1 Requirements

Requirements are used to ensure that the system fulfills a specific functionality or behavior. The same applies with models, where requirements are used to state important functions. These are then used to create CQAS and acceptance tests for the model, which helps improve the V&V process. An effective way of formulating the requirements is to create user stories. User stories follow the form of who wants what and why, most commonly "as a *<role>* I want *<functionality>* because *<reason>*". They provide a compact way of describing the desired functionality, who benefits from it and its purpose.

4.2.2 Concrete Quality Attribute Scenario

Quality attributes are used in software engineering to describe a measurable or testable property of a system [35]. A quality attribute is used to indicate how well a system fulfills its requirements from stakeholders. CQASs are used for each requirement to further explain where the source of the data comes from, how it interacts with the system and what is expected from the interaction. This allows for even more specific information about what is expected from a requirement than just having it as a user story. It also allows for validation of models, where real-life/system data may not exist, as CQASs can be used to compare to expectations obtained from the earlier stated assumptions. A CQAS is divided into six parts, each describing one section of the requirement, these can be seen in Table 4.1.

Stimulus	An event that arrives to the model
Source of stimulus	Where the stimulus came from
Response	How the model is expected to respond to the stimulus
Response measure	Measure of the response
Artifact	The part of the system that the requirement applies to
Environment	The condition of the model when the stimulus happen

Table 4.1: Table for defining a Concrete Quality Attribute Scenario.

4.2.3 Acceptance test

Acceptance tests are high-level descriptions of some desired system behavior or functionality. These tests are designed in relation to a user story (a requirement) to establish the behavior of the system in regard to conditions, input, and output data. The main purpose of using acceptance tests is to provide a way of determining whether a requirement is satisfied. The tests focus mainly on what functionality to implement, but not how it is achieved. They are usually designed in coordination between different project roles (e.g., customer, developer, tester etc.) to include contributions from different stakeholders.

A simplified version of the acceptance test can be provided in pseudocode to allow

for a coherent overview of what is required for the test. In addition, a description of the acceptance test should be provided to clarify any ambiguities. During the creation of the acceptance tests, an OpenSCENARIO file or an OpenDRIVE file should be created or specified, so that it can be used for the later testing.

4.3 Model development

In this step, the model is developed and the functionality described by the previously elicited requirements is implemented. Unit tests are added to test smaller individual parts of the model. Additionally, the execution of both unit tests and acceptance tests should be automated in order to enable efficient and iterative confirmation on whether tests are passed. This step is part of the ATDD process of implementing functionality that passes the acceptance tests defined in step 2 in the flowchart in Figure 4.1.

4.4 Calibration, Verification, and Validation

The calibration, verification, and validation step is a combination of several steps inspired by Barceló [13], [14] and ATDD [37], that are further explained below. The meaning of these steps is to test the model developed in step 3 based on the requirements and acceptance test defined in step 2. The model should also be validated towards the assumptions and acceptance criteria defined in step 1. These steps can be clearly seen in the flowchart in Figure 4.1.

4.4.1 Iterative process

The calibration, verification, and validation is an iterative process that will most likely occur several times during the development of a model. It is during this part of the development that the model is tested with the acceptance tests to ensure that the requirements have been met, thus ensuring that the model is calibrated. These iterations include further connection to stakeholders to ensure that no new requirements have occurred or are needed for the model to behave accurately. The iterations also include the validation where the model is compared to the expected behavior and the stated acceptance criteria. The purpose of using an iterative process is not to document every iteration and record what was changed, but to continuously improve the model.

Each iteration contains five steps, these are:

1. Create unit tests
2. Run tests
3. Perform refactoring and calibration
4. Reconnect with stakeholders
5. Empirical and visual validation

4.4.2 Create unit tests

Barceló [14] pinpoints the importance of unit tests to verify the simulation model. Unit tests are used to test small, individual units of the model. The goal is to isolate a part of the model and identify faults in earlier stages of the model development cycle. Unit tests can also be used for regression testing during refactoring, ensuring that previously working modules are still working correctly. Creating unit tests is a stepping stone to verify smaller modules of a model and subsequently build larger and more complex tests.

4.4.3 Run tests

During this step, all acceptance and unit tests are performed to test that the current model has the expected behavior based on the requirements stated in the earlier step. If any of the tests does not pass, one should revert to the develop-model step and solve the problem that results in the test failing.

4.4.4 Perform refactoring and calibration

If the acceptance and unit test has passed, the code of the model can be refactored to ensure that it is maintainable and understandable. Examples of this can be breaking long functions into several smaller functions or removing duplicate code blocks. Refactoring also involves modifying or improving other artifacts, such as clarifying the assumptions document. Calibration of the model means that parameters are adjusted to generate behaviors that are closer to the expected behavior. Additionally, refactoring may cover performance optimizations, as esmini can run simulations faster than real-time and is therefore dependent on optimized code. If any refactoring or calibration has been performed, the acceptance and unit tests should be performed once again to ensure that the refactoring has not introduced any unwanted behavior into the model.

4.4.5 Reconnect with stakeholders

Reconnecting with the stakeholders iteratively during model development is essential to make sure the right model is being created. This includes analyzing if the current result satisfies the stakeholders, thus checking the validity, but also reiterating the stated requirements and potentially add new requirements or modify existing ones. Thus, stakeholders are continuously involved in the development process, which leads to a higher likelihood of eliciting requirements that represent the desired model functionality.

4.4.6 Empirical and visual validation

Validation should be performed on the model as the last step, including methods such as statistical comparison, empirical analysis, or visual inspection can be used to determine the model's accuracy. This should be used to the maximum extent possible. However, the use of statistical and empirical methods can be difficult to

use on certain models in esmini, as esmini is a deterministic simulator. If the model reaches the acceptance criteria, it can be seen as validated. If not, revert to the assumption document to find why the model behaves unexpectedly and use the found information to improve the model in the next iteration.

Data extraction: To validate the model, data needs to be extracted so that it can be compared to the expected behavior or real life/system data. This can be performed in different ways throughout the validation process, either manually or with automated tools if possible. For an example of how to acquire the needed data, see Section 5.4.3.

Empirical: Calculated data extracted from the model should be compared to the expected behavior or real system data to determine how accurate the model is, thus validating the behavior of it.

Visual: Data from the model can be validated through visual comparison between the model's data and the expected behavior. An example of such validation techniques are Turing test or animation [15], where the expert visually inspects the model's behavior and deem it reasonable or not.

5
Framework Application Example

This section provides an illustrative example of how the framework can be used on an actual model.

The case the developed model, LIRM, aims to solve is the implementation of lane independent routing in esmini, since as of now esmini can not find routes to a target that does not exist in a road connected to the lane of the starting position. The lane independent routing model should be able to find the route to target in all lanes. To solve this, the model must also simulate lane changes so that the modeled vehicle can change lane if necessary. A more detailed description of the case can be found in Appendix A.

5.1 Assumptions document

The LIRM that is specified here aims to solve the problems and objectives stated in the case description from Volvo Cars A. In short, LIRM aims to achieve the independent lane routing functionality that allows navigation to target waypoints where lane changes are required. A waypoint is defined as a position in the road network, which specifies the road ID, lane ID and the offset road length from the starting point of the road (s-pos). The validation of the model helps to ensure that the model functionality is as close to the desired functionality as possible.

5.1.1 Model functionality

- Cars defined can find their way through a road network without having to be placed in the correct lane.
- A route following controller has to be created, implementing the routing and modeling of lane changes.
- The model should handle lane changing with some degree of realism, for example, maybe not change lane 2 meters before an exit on a highway.
- The routing should support three different routing strategies as defined in OpenSCENARIO, shortest, fastest and least number of intersections.

5.1.2 Model flowchart

To help with the development of the LIRM, a simplified flowchart over the LIRM's expected parts were created. The flowchart can be seen in Figure 5.1. The impor-

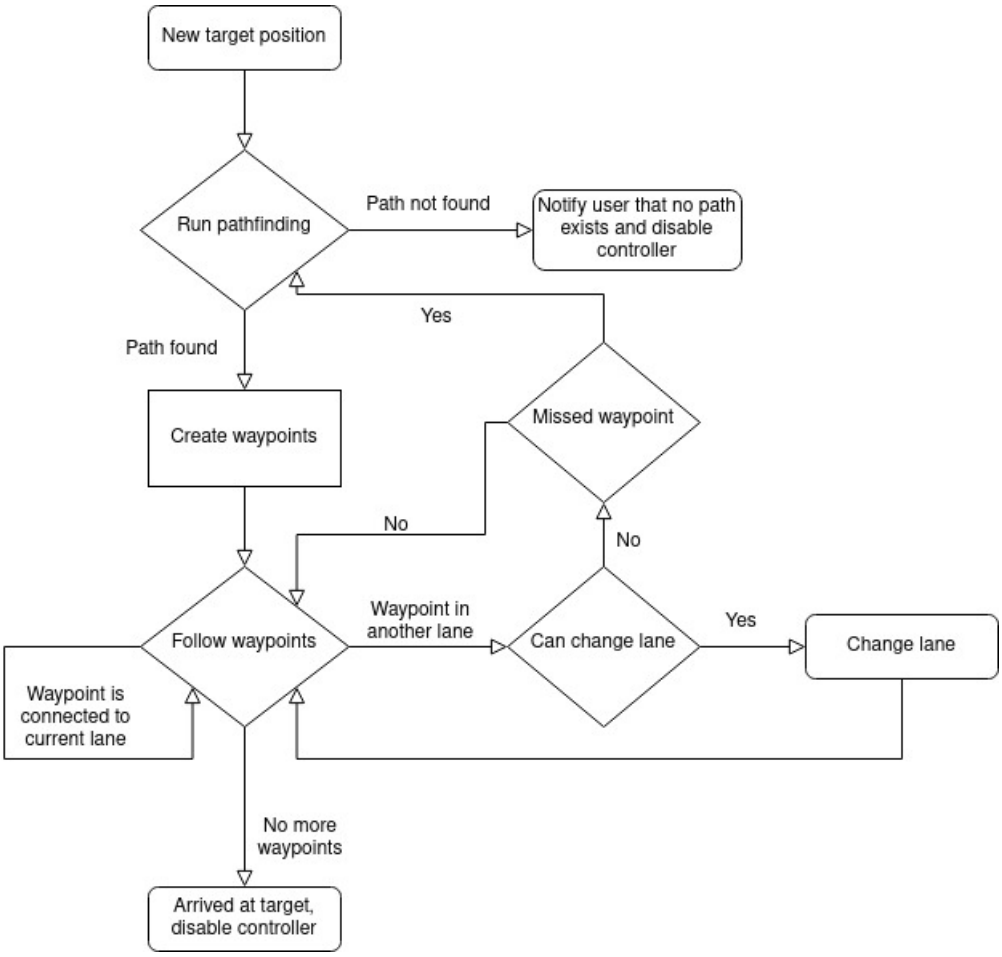


Figure 5.1: A simplified flowchart of the model for the case.

tant modules in the LIRM are pathfinding and waypoint follower. The flowchart also display how lane changes are performed, and how missed waypoints should be handled.

5.1.3 Model subsystems

The case needs the LIRM to be divided into two subsystems, one for the pathfinding and one for the waypoint following that is responsible for lane changes. The pathfinding result is used to create a list of waypoints that the waypoint follower can use.

5.1.4 Simplifying assumptions

- Road networks are not infinite / have some sort of limit on the size
- Only checks first and last lane section, as a road can not continue on after a junction by the OpenDRIVE standard, as virtual junctions are not implemented
  - The assumption is that connections of roads can only happen at the start or end of a road. For example, in a three-way intersection, no road can

continue through, all roads connected to the "junction/intersection" are different
- All lanes are always seen as switchable
  - On a specific road, we assume that we can always change lane to another lane in the same direction as we are driving (assuming that it is drivable)
- For fastest routing strategy
  - Currently, the average speed of the entire road is used to calculate the travel time (seconds) that is used as weight for Dijkstra's algorithm
  - If the road does not have a specified speed limit, the road type will be checked
    - If road type is motorway → speed limit = 90 km/h
    - If road type is rural → speed limit = 70 km/h
    - If road type is town → speed limit = 50 km/h
    - If road type is low speed → speed limit = 30 km/h
    - Else we assume that the road type is rural → 70 km/h
- Lane changes should have a degree of realism
  - a lane change should be performed within approximately 5 seconds
  - no lane change should be performed if another car exists in the target lane (avoid collisions) within 10 meters.

5.1.5 Model limitations

- The model, LIRM, is developed after the rules and limitations stated in OpenDRIVE v1.7.0. and OpenSCENARIO v1.1.1.
- For fastest routing strategy
  - All lanes in a specific road have the same speed limit (as lane dependent speed limits are not implemented yet in esmini)
- The target lane must be defined as drivable for the algorithm to find it (there are different kinds of lanes in OpenDRIVE, e.g., borders, sidewalks, driving, biking, etc.)
  - This applies to all lanes that are in the path
- Lanes can't connect with lanes with different heading (no U-turn)
- Driving in the opposite direction of the road is not allowed

5.1.6 Model input data summary

The input for LIRM contains the following data, see Figure 2.1 for visualization.
- Start position → (s-pos, offset, lane ID, road ID)
- Target position → (s-pos, offset, lane ID, road ID)
  - road ID → ID for the specific road that the position is placed on
  - s-pos → longitudinal position, the length (in meters) along the specific road.
  - lane ID → ID for the specific lane that the position is placed in (if omitted, reference line is used (0))

  - offset → offset relative to current lane ID
- Routing strategy → Shortest | Fastest | Least number of intersections
- Road network → OpenDRIVE road network (.xodr file)
  - Translated to esmini data structures (road, junction, lane, lane section)
- Scenario file → OpenSCENARIO
  - Contains waypoints, vehicle start pos, vehicles, etc.

5.1.7 Important sources for information

- OpenDRIVE [27]
  - Provides information related to road networks. This includes lanes, roads, junctions, speeds, road marks etc.
- OpenSCENARIO [28]
  - Provides information related to scenarios that are performed in a road network. This includes storyboards, maneuvers, events, actions, triggers, entities etc.
- Case definition from Volvo, seen in Appendix A
  - Any information related to the case, assumptions or miscellaneous information about esmini.

5.1.8 Acceptance criteria

The acceptance criteria of LIRM are:

- All acceptance tests should pass.
- The model should find the shortest, fastest and minimum number of intersection routes through a road network and not differ greatly from the optimal routes.
- The model should perform lane changes that can be deemed realistic.

5.1.9 Assumption validation

The assumptions document was sent to the thesis supervisor at Volvo Cars, and a confirmation that the assumptions document was acceptable and correct was acquired after a walkthrough.

5.2 Requirements, CQAS and Acceptance Tests

In this section, the requirements, CQAS and acceptance tests for the model, LIRM, are stated.

5.2.1 Requirements

Shown in Table 5.1 is a selection of the requirements that were created for LIRM, the entire list of requirements can be found in Appendix B.

Requirement	As a <role>, I want <functionality>because <reason>.
1	As a developer, I want the algorithm to calculate a path within a reasonable amount of time for the given size of OpenDRIVE road network, as it should not slow down the rest of the simulation.
2	As a developer, I want the algorithm to find the expected/optimal path for a given route strategy (the strategy that the pathfinder should use), since it should be close to an ideal pathfinder.
3	As a developer, I want the modeled vehicle to be able to reach the target waypoint, in order to follow the path created by the lane independent pathfinder.

Table 5.1: A selection of requirement for the developed model.

5.2.2 Concrete quality attribute scenario

The concrete quality attribute scenarios for the requirements found in Table 5.1 can be found in Table 5.2, Table 5.3 and Table 5.4

Stimulus	A valid target position
Source of stimulus	User or read from the OpenSCENARIO file
Response	A path, (sequence of roads to traverse)
Response measure	How long it takes to find / calculate the path
Artifact	The lane independent path finding algorithm
Environment	Runtime, after a new target as been received

Table 5.2: CQAS for requirement 1

Stimulus	A valid target
Source of stimulus	User or read from the OpenSCENARIO file
Response	The expected/optimal path, (sequence of roads)
Response measure	The found path
Artifact	The lane independent path finding algorithm
Environment	Runtime, after a new target as been received

Table 5.3: CQAS for requirement 2

Stimulus	A list of waypoints
Source of stimulus	After path has been found, waypoints has been created
Response	The modeled vehicle drives to target.
Response measure	If the modeled vehicle has reached target
Artifact	Lane change controller
Environment	Runtime, after waypoints has been created

Table 5.4: CQAS for requirement 3

5.2.3 Acceptance tests

In this section, the acceptance tests for the requirements in Table 5.1 can be seen.

Acceptance test for requirement 1.
Uses the medium road network, multi_intersections.xodr, seen in Figure 5.2. Calculates the time to find a path through the network and checks if it is less than 5 ms. The pseudocode for the test can be seen in Code 1.

```
Start position: s=100, t=0, road id = 202, lane id = 2
Target position: s=20, t=0, road id = 209, lane id = 1
Driving direction: opposite of road
    startTime = save current time
    path = result from pathfinder function
    endTime = save current time
Verify that the returned path list is not empty
Verify that the last node in path has road id 209
calcTime = endTime - startTime
Verify that calcTime is less than < 5ms
```

Code 1: Acceptance test for requirement 1 in pseudocode.

Acceptance test for requirement 2.
The expected path can be acquired by manually tracing and finding the optimal/expected path through the OpenDRIVE (.xodr) road network, multi_intersections.xodr, seen in Figure 5.2. The pseudocode for the acceptance test can be found in Code 2.
This test should be repeated for each route strategy, SHORTEST, FASTEST and LEAST INTERSECTIONS.

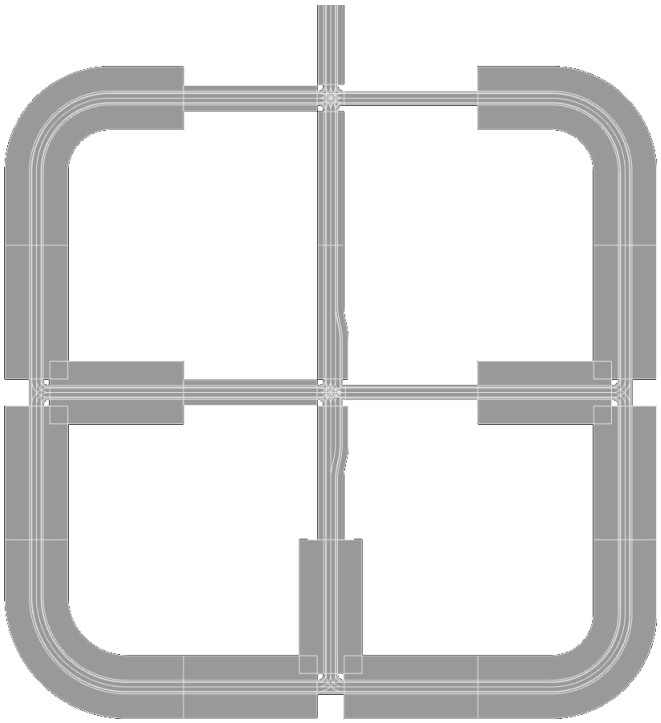


Figure 5.2: An overview of the road network used for the acceptance tests 1&2.

```
1.  Trace the OpenDRIVE road network and calculate the
    expected route based on each route strategy from a start
    position to a target position for 5 routes. This step
    could be automated in the future, but was performed manually
    in this case due to time constraints.
2.  Enable path logger in the simulation model.
3.  Run the simulation model, with the start and target positions
    of the 5 routes.
        Collect the calculated route for each route from the model.
4.  Use the expected route and the collected calculate route
        Visually inspect the routes and compare them to determine
        if the calculated path takes the same route as the expected.
```

Code 2: Acceptance test for requirement 2 in pseudocode.

Acceptance test for requirement 3.
Create a scenario file with the start and target positions, with a vehicle that has the controller activated. Use the small OpenDRIVE road network seen in Figure 5.3. The pseudocode for the acceptance test can be found in Code 3.

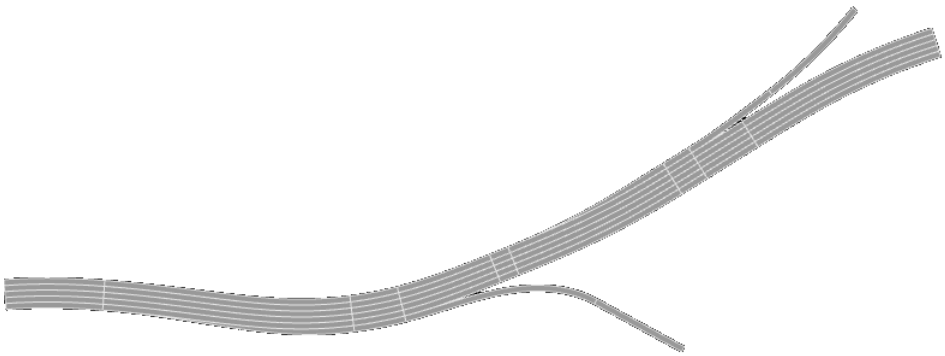


Figure 5.3: An overview of the road network used for the acceptance test 3.

```
Start position: s=10, t=0, road id = 0, lane id = -1
Target position: s=20, t=0, road id = 2, lane id = -1
Driving direction: in road
Run the simulation
Acquire the last position of the vehicle
Verify that last position is the same as target
```

Code 3: Acceptance test for requirement 3 in pseudocode.

5.3 Model development

In this step, LIRM is developed, and the functionality described by the previously elicited requirements is implemented. The LIRM is seen in Figure 5.1. Unit tests are added to test smaller individual parts of the model. Additionally, both the unit tests and acceptance tests should be automated in order to enable efficient and iterative confirmation on whether tests are passed. This step is part of the ATDD process of implementing functionality that passes the acceptance tests defined in step 2 in Figure 4.1. For the technical explanation of the developed model, see Section 3.4.

5.4 Iterations

The development of LIRM followed the framework's process of iterations. However, as the number of iterations performed during the actual development are more than would be feasible to include in the example of use, the choice fell on displaying the first, second and the final iteration. This as these would display the important functions of the iterative process and how the final empirical and visual validation could be performed. By just showing these three iterations, the example should be easier to follow than several iterations with no meaningful information. As many of the iterations performed during the development would end when running the tests and seeing that one or more tests failed.

5.4.1 First Iteration

For the first iteration, the development of LIRM began. At this stage, the model had three acceptance tests, shown in Table 5.1. The first iteration of the development focused primarily on implementing the lane independent pathfinder, as the lane change controller needed the lane independent pathfinder to work before it could be developed.

Three unit tests were created to test the average speed calculations on a road. The tests were used to make sure the calculated average speed was correct for various roads: roads without a defined speed, roads with a defined speed and roads with more than one road type.

After the first version of LIRM was developed, the test was run to see if the model worked as intended, the results can be seen in Table 5.5. The acceptance test for requirement 3, was not tested as the development of the lane change controller had not yet begun.

Req.	Test type	Test name	Result
1	Acceptance	FindPathTimeMedium	Passed
1	Acceptance	FindPathTimeLarge	Passed
2	Acceptance	FindPathShortest	Passed
2	Acceptance	FindPathFastest	Failed
2	Acceptance	FindPathMinIntersections	Passed
-	Unit	CalcAverageSpeedForRoadsWithoutSpeed	Passed
-	Unit	CalcAverageSpeedForRoadsWithDefinedSpeed	Passed
-	Unit	CalcAverageSpeedForTwoRoadTypes	Passed

Table 5.5: The test results from running the acceptance and unit tests during the first iteration.

As seen in Table 5.5, the acceptance test FindPathMinFastest failed. The reason for the failure was that no path was found from start to target, due to the last road not being visited by the pathfinder. In this case, the framework directed the process back to the development of the model in order to improve the model and make sure all tests passed before moving on to the refactoring step.

5.4.2 Second Iteration

The second iteration focused on solving the problem that caused the test FindPathMinFastest to fail. Due to this, no new unit test or functionality was added during this iteration. The solution to make test pass was to not stop the pathfinding on the last connecting road to target, but also add it to the visited list so that it is included when backtracking the fastest path.

Req.	Test type	Test name	Result
1	Acceptance	FindPathTimeMedium	Passed
1	Acceptance	FindPathTimeLarge	Passed
2	Acceptance	FindPathShortest	Passed
2	Acceptance	FindPathFastest	Passed
2	Acceptance	FindPathMinIntersections	Passed
-	Unit	CalcAverageSpeedForRoadsWithoutSpeed	Passed
-	Unit	CalcAverageSpeedForRoadsWithDefinedSpeed	Passed
-	Unit	CalcAverageSpeedForTwoRoadTypes	Passed

Table 5.6: The test results from running the acceptance and unit tests during the second iteration.

The failing test in the previous iteration, FindPathMinFastest, is now fixed and pass in this iteration, as seen in Table 5.6.

After the test passed, the code was studied to see if there were any need for refactoring to ensure that the code was understandable and maintainable. In this iteration, a code block testing if the start and target position was valid was found duplicated on two locations. To remove the duplicate code and increase maintainability, the code block was extracted to a function.

After the refactoring was performed, the tests were run again to ensure that the refactoring did not break any functionality. Since all tests passed, the next step was to reconnect with the stakeholders and see if the current version fulfilled their expectations. During the reconnection with stakeholders, a few new requirements were stated, both for the lane independent pathfinder and the lane change controller. These requirements were adapted to CQAS and then in the acceptance tests.

5.4.3 Final Iteration

At the final iteration, the LIRM's three requirements had been extended with an additional eight, resulting in a total of twelve requirements, seen in Table 5.7. The requirements and their corresponding CQAS and acceptance tests can also be found in Appendix B.
The final iteration contained 32 total tests, compared to the first and second iteration, which had 7 tests. As this was the final iteration, no refactoring was performed, nor were any requirements added to the LIRM's behavior.

In addition to the increased number of total tests, empirical and visual validation were added to some acceptance tests to support their validity. This was especially important for lane changes, as empirical data was used to compare the observed lane change data to expected behavior in order to achieve the required degree of realism. Likewise, comparing the behavior through visual validation was done to ensure a behavior that was visually adequate to what was expected.

Req.	As a <role>, I want <functionality>because <reason>.
1	As a developer, I want the algorithm to calculate a path within a reasonable amount of time for the given size of OpenDRIVE road network, as it should not slow down the rest of the simulation.
2	As a developer, I want the algorithm to find the expected/optimal path for a given route strategy, since it should be close to an ideal pathfinder.
3	As a developer, I want the modeled vehicle to be able to reach the target waypoint, in order to follow the path created by the lane independent pathfinder.
4	As a developer, I want the algorithm to be able to find a path (sequence of roads to traverse) between the vehicle position to a valid target, so that the simulated vehicle can reach the target.
5	As a stakeholder, I want the algorithm to find the path based on the routing strategy “shortest”, as it is defined in the OpenSCENARIO specification.
6	As a stakeholder, I want the algorithm to find the path based on the routing strategy “fastest”, as it is defined in the OpenSCENARIO specification.
7	As a stakeholder, I want the algorithm to find the path based on the routing strategy “the least number of intersections”, as it is defined in the OpenSCENARIO specification.
8	As a developer, I don’t want esmini to crash if the algorithm can’t find a valid path, as I want to be informed of this instead and keep esmini running.
9	As a developer, I want the functionality to translate paths to waypoints, so that the model can follow them.
10	As a developer, I want the modeled vehicle to be able to change lane, in order to follow the path created by the lane independent pathfinder.
11	As a tester, I don’t want the lane change to happen if it results in a collision, as it would be “unrealistic”.
12	As a tester, I want the lane change to have a degree of realism, since the model could be used for vehicle testing.

Table 5.7: All requirements for the developed model in the final iteration.

Requirement 2

Requirement 2 states that pathfinder should find the expected/optimal path between two positions, so far this has only been tested as a part of the automated tests, this however is not enough to validate the pathfinder. To increase the validation and make sure that the pathfinder behaves as wanted, visual and empirical validation was performed. The OpenDRIVE road network used for this validation can be found on Github.[1] The road network that was used for this test is not quite realistic, however it was chosen as it was determined to be the most suitable network for testing all routes strategies on, due to it having several junctions and roads with different speeds and lengths.

Visual validation: The visual validation for the pathfinder is performed by first manually trace the expected path between two positions and then run the pathfinder to see if it takes the expected path. This was performed for five paths for each of the route strategies that the pathfinder supports.

Expected data: The expected data was traced on the OpenDRIVE road network, after what would be the expected path after following the behavior of an "average law-abiding" driver with respect to the limitations of the model, for example no use of U-turns or other stated driving behaviors. The OpenDRIVE road networks were opened in a 3D-viewer called OpenDRIVE Viewer[2], which allows the user to see and interact with the road network, which was necessary for extracting the expected path. The expected paths were then drawn on a map of the road network, as this would be used to validate if the calculated path from the pathfinder was the same as the expected.

Calculated data: The calculated data was extracted from LIRM by implementing a test which logs all waypoints *(road-ID, lane-ID, S-position)* that are created by the pathfinder for a given path. This data was written to a .csv file so that it would be able to used to then visualize the calculated path similarly as the expected path. The waypoints were then manually marked on a map of the road network and the path was traced between them.

Validation: To perform the validation, five routes were created by randomly selecting a start and a target position on the map, these can be seen in Table 5.8.

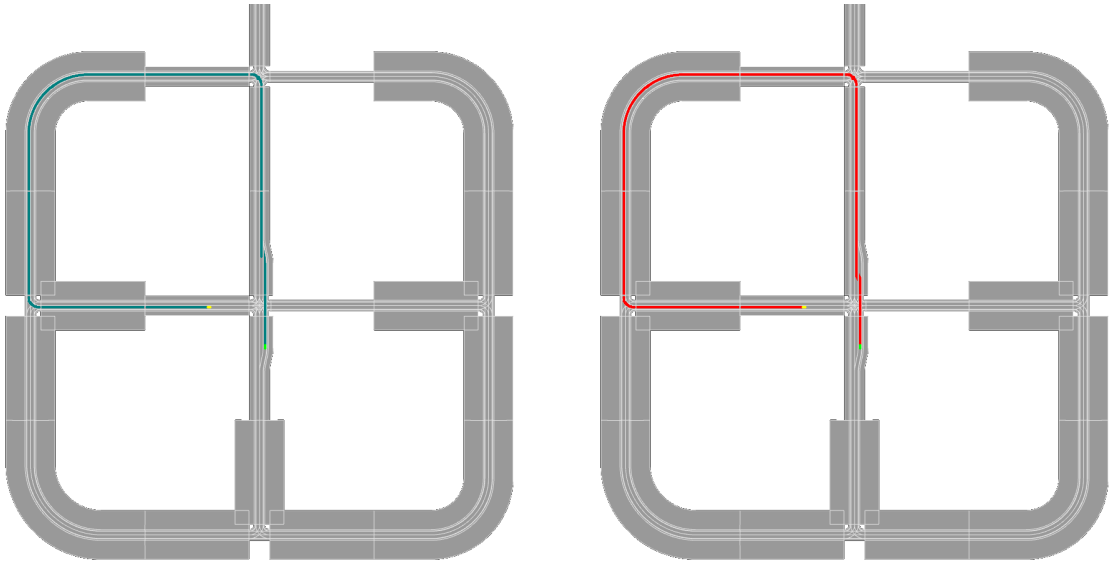
Criteria: The criteria of requirement 2 is that the calculated path should be the same as the expected path. This could be further broken down to that the calculated path should take the same roads as the expected path to reach the target. However, as the calculated path is calculated on a road network with no other vehicles than the simulated one, and as it implements the default driver model in esmini, it can not be expected to take the same decisions regarding when and where lane changes are performed, as the average driver would.

[1] https://github.com/esmini/esmini/blob/follow_routes/EnvironmentSimulator/Unittest/xodr/multi_intersections_changed_speeds.xodr

[2] https://odrviewer.io

Route	Start	Target
1	(266, 1, 29)	(275, 1, 55)
2	(202, 2, 33)	(196, 1, 49)
3	(197, -1, 52)	(267, 1, 32)
4	(227, 1, 55)	(242, -1, 50)
5	(242, 1, 50)	(197, -1, 51)

Table 5.8: Start and target positions of the routes used in validation, written in the format (road-ID, lane-ID, S-position).

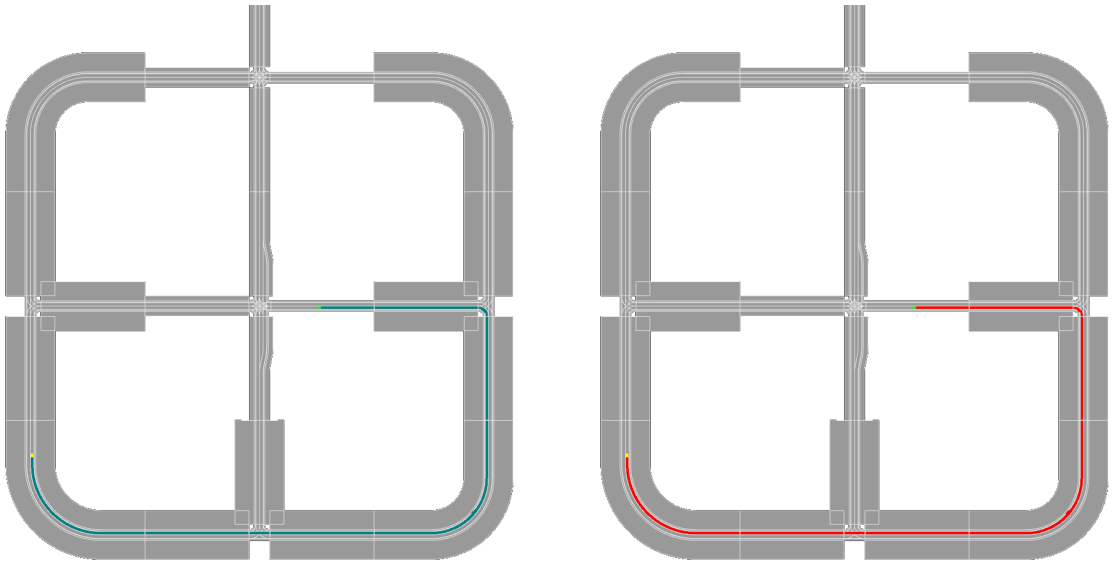


(a) Calculated path **(b)** Expected path

Figure 5.4: Calculated and expected path for route 2, using route strategy: shortest.

Route strategy, Shortest: The visual validation for the route strategy, shortest for route 2, can be seen in Figure 5.4. As can be seen in the figure, there are not many clear differentials between the calculated and expected path, this is due to that the road network is mostly made up of one lane roads. However, in the middle section a small difference can be seen between the calculated and expected path, this as the expected path that follows an average driver would change lane before the end of the lane to reduce the chance of accidents or the need of stopping. But as stated in the criteria in the previous paragraph, the requirement does to not require the calculated path to be exactly the same as the expected, so the model fulfills it for this route. All other routes were also tested and visually inspected. The model fulfilled the criteria for each route, which validates that the model behaves as expected for the route strategy shortest.

Route strategy, Fastest: The visual validation for the route strategy, fastest for route 3, can be seen in Figure 5.5. As show in the figure, the calculated path and

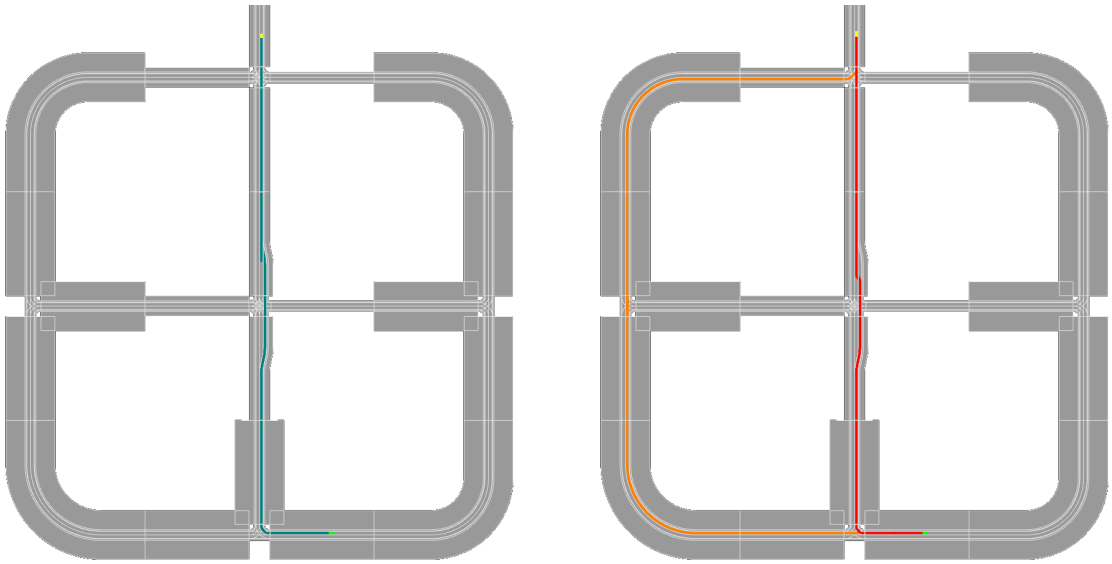


(a) Calculated path **(b)** Expected path

Figure 5.5: Calculated and expected path for route 3, using route strategy: fastest.

the expected path are practically the same. The single lane layout of the expected path leaves no room for any deviations regarding lane changes. In order to follow the fastest path, the calculated path has to be virtually identical to the expected path in this case. The model fulfilled the criteria on the other four routes as well, which validates that the model behaves as expected for the route strategy fastest.

Route strategy, Least intersections: The visual validation for the route strategy, least intersections for route 4, can be seen in Figure 5.6. The expected path on for this route is different from the other paths, as there are two valid paths to the target. These are displayed with the red and orange lines. The expected path is more likely the orange path, as an average driver would in most cases take the fastest path if two were presented. However, as the model can not use more than one route strategy at once, the calculated path can be either of these while still fulfilling the requirement. As seen, the calculated paths follow the red expected path. As it can be seen there are not any significant deviations between the expected and calculated path, this is due to what have been mentioned previously for the route strategy shortest. The model fulfilled the criteria on the other four routes as well, which validates that the model behaves as expected for the route strategy least intersections.



(a) Calculated path **(b)** Expected path

Figure 5.6: Calculated and expected path for route 4, using route strategy: least intersections.

Requirement 12

Requirement 12 states that a lane change should have a degree of realism, as LIRM could be used for testing vehicle software in the future, therefore it is important that the performed lane changes behaves realistic, so that tested software is tested correctly. This could not be performed with the automated tests, as it needed better validation than what could be performed with the tests. So to validate this requirement, the choice fell on empirical validation.

Empirical validation: To validate requirement 12 for the model, expected and calculated data can be compared against each other to see if the calculated data is within the range of what the requirement specifies to be validated. The expected data was collected through observations, in other words through empirical research on lane changes. This comparison can be performed in several ways, this example will compare the data by displaying it in graphs.

Expected data: The expected data regarding lane changes was obtained from Volvo Cars, containing information about a vehicle's longitudinal and latitudinal changes and the velocity of the vehicle over time. This data was then used to extract an average time for the duration of a lane change. For the provided data seen in the Figure 5.7, the average lane change time was approximately 5 seconds at any given speed, the majority of the recorded lane changes followed the form of a sinusoidal curve and the average distance traveled during a lane change was between 100 and 150 meters.

Calculated data: The calculated data for two lane changes was gathered by run-

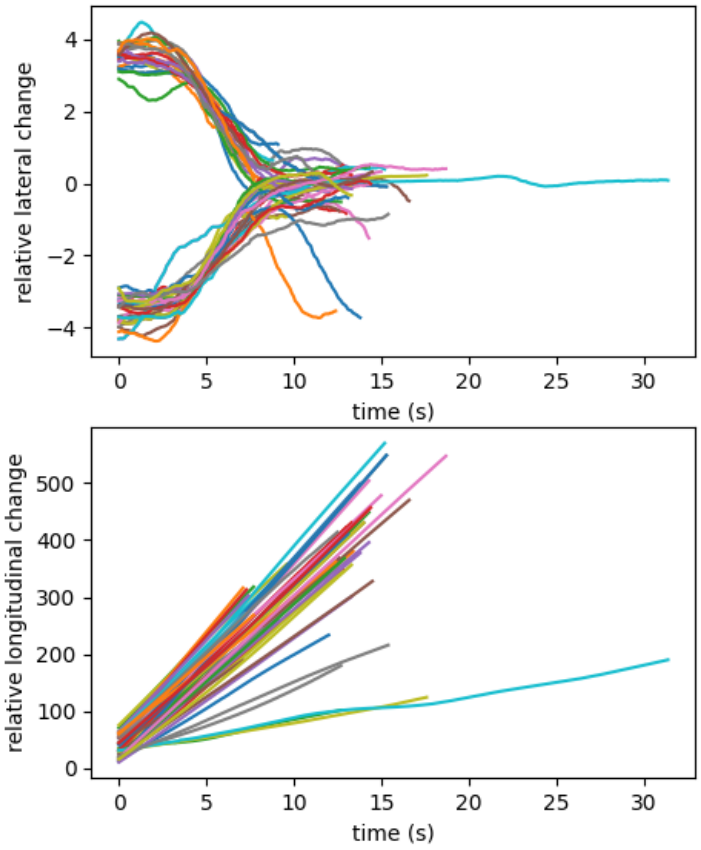


Figure 5.7: Graphs of lateral and longitudinal positions from expected data regarding a lane change. The upper graph displays relative lateral position (m) over time (s). The bottom graph displays relative longitudinal position (m) over time (s).

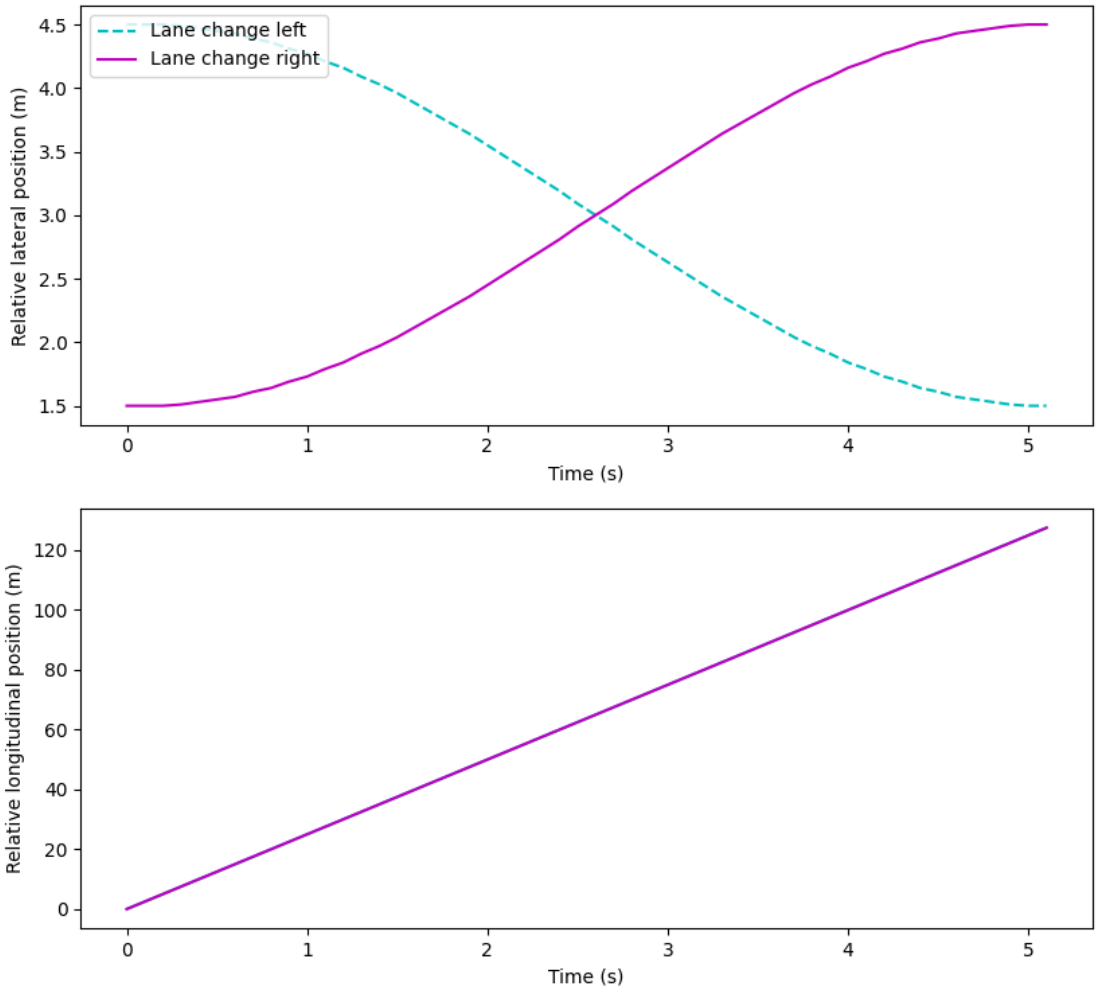


Figure 5.8: Graphs of lateral and longitudinal positions from calculated data in esmini. The upper graph displays relative lateral position (m) over time (s). The bottom graph displays relative longitudinal position (m) over time (s).

ning LIRM and recording the lateral change in position of the vehicle and the longitudinal change of the vehicle during a lane change. The lateral value is relative to the reference line of the road, increasing to the right and decreasing to the left. The collected data can be seen in Figure 5.8. From the calculated data, it can be seen that a lane change takes approximately 5 seconds to complete, as seen in the upper graph of Figure 5.8. The solid line represents a lane change in the right direction, while the dotted line represents a lane change in the left direction. In the same graph, it can also be seen that the lateral movement during a lane change follows a sinusoidal curve. In the lower graph, it can be seen that the longitudinal movement of the vehicle performing the lane change is approximately 125 meters. As this is a simulation model that does not have any variation applied, the calculated data will always be the same if no parameters were changed. This is what is causing the lower graph to only display one line, as both lines are identical to each other.

Evaluation: By comparing the data of the real lane changes, (the expected data)

with LIRM's performed lane changes (calculated data) it is clearly seen in the graphs that the model correspond quite accurately to how a real lane change is performed. Since the overall shape of the modeled lane change follows a similar sinusoidal curve and takes approximately 5 seconds to complete. This verifies and validates that the model fulfill requirement 12, which states that the modeled lane changes should have a degree of realism.

6
Results

In this chapter, the results collected during the thesis will be presented, in order to answer the research questions that were stated in Section 1.3. The first section will describe the steps needed to create a framework for V&V, while the second section will present the result from the evaluation of the framework.

6.1 Developing a framework

In order to develop a framework for V&V in regard to simulation models, a set of procedures and methods were used. The first step was to choose an appropriate research method for the task at hand. In the case of this thesis, the method it fell upon was Design Science Research Process, as seen in Section 3.1. It was found appropriate as it involves an agile/iterative development process, which allowed for constant improvements of the framework. After the research methodology was chosen, the next stage was to begin to gather and build knowledge about the systems that the framework should be applied upon, as well as gaining knowledge of current methodologies. This is also why DSRP was found to be useful for the development of the framework, as the first two steps of DSRP are to identify the problem and define a solution. The next step is to use the gained knowledge about the system and existing methodologies to design and develop the framework. This step was performed iteratively with constant deployment on a specific case to gain feedback on what could be improved, this iterative process can also be found in DSRP.

The developed framework and the processes used within it can be found in Chapter 4. The desk research, which contains descriptions of existing frameworks and methodologies, is seen in Chapter 2.

6.2 Evaluation of framework

In this section, three different methods will be used to determine how effective and useful the framework.

6.2.1 Feedback

Feedback regarding the framework was collected by interviewing esmini developers at Volvo Cars. Four questions were asked, these were:

1. How did you perceive the structure of the framework, were all steps easily understandable and if not, what steps were more difficult to understand?
2. Did the application example of the framework provide useful insight on how the framework can be applied to models in esmini?
3. From your experience, how useful do you think the framework would be in practice?
4. Is there any step of the framework process that you would consider changing or add to the framework?

The interviewees perceived the framework structure to generally be easy to understand. For instance, the various flowcharts were much appreciated. There were some questions on how useful empirical validation is in esmini, since esmini is a pure and deterministic simulated environment, which could increase the difficulty of applying empirical validation. However, they all agreed that for some models, the empirical validation can be valuable. By some models, they refer to models that implement certain realistic features that needs to be validated against real data. One example of such a model is the lane change validation seen in Section 5.4.3, where real lane change data was analyzed in order to validate the lane changes performed by the model.

The application example of the framework proved to be a very appreciated visualization of how the framework can be applied. The example provided the necessary amount of information to be easily understandable by different roles, since it showed how the overall development should be performed and how requirements could be added throughout the process.

The interviewees agreed that the framework would be useful as a starting point and guideline for how the development of larger models in esmini should be performed to ensure that the models are verified and validated. On the other hand, for smaller models or functions in esmini, they felt that the framework might be too time-consuming or excessive to use. A benefit that was mentioned was that the use of a framework such as this one would force the developers to actually validate the models and document assumptions. The integration of the framework would be quite seamless as several steps of the framework are used similarly to the current development process, for example unit testing and some degree of requirement usage.

None of the interviewees had any changes that they wanted to incorporate into the framework, neither were there any suggested additions to the framework.

6.2.2 Simulated experiment

The framework was quantitatively evaluated through simulated experiments in the form of fault injections and removal of parts of the framework. The experiments were performed to ensure that the framework detects any introduced faults into the model and that some parts of the framework are essential to retain a complete V&V process. The following examples show how the framework handles models that

should not pass the validation process.

Fault injection

To evaluate the framework, three faults were injected into the code of the model, LIRM. The injected faults can be seen in Table 6.1. These faults were chosen due to them affecting one or more requirements. After the faults were injected, the framework process was performed on the model, to see if it would detect the injected faults.

Fault	Description
1	Incorrect road weight calculation (random value between 10-100 added to weight when calculating with route strategy shortest)
2	Wrong nodes to positions translation (the lane ID was acquired from the incorrect node)
3	Inaccurate lane change behavior (duration and shape incorrect)

Table 6.1: A description of the injected faults.

Automated tests

Fault 1 was detected by the automated tests, the acceptance test "FindPathShortest" failed, as well as the unit test "CalcWeightShortest". The tests indicate that something is incorrect for the weight calculation for the route strategy shortest, alerting what functions are misbehaving. Fault 2 was detected by the automated tests, where nine tests failed after the fault was injected, strongly indicating that something is not behaving as expected. From the tests that failed, "FollowRouteMedium" and "CreateWaypointsMedium" being two of them, it was quite clear that the problem which caused the tests to fail existed in the waypoint creator function. However, Fault 3 was not detected by the automated tests.

Empirical and visual validation

Fault 3 was detected during the last step of the framework, during the verification of requirement 12, lane changes should a degree of realism. When the calculated data was collected from LIRM, it was clearly seen that the calculated data, Figure 6.1, did not correspond accurately to the expected data, seen in Figure 5.7. It was shown that the LIRM's lane changes did not have the degree of realism needed, as it did not have the proper shape nor a duration that was close to the expected data. Thereby, showing that something was incorrect regarding how the lane changes were modeled in the model.

Removing parts of framework

To evaluate if different parts of the framework are useful, they will be removed and then the process of the framework will be simulated after the example of use seen in Chapter 5. This means that the entire framework process will not be reapplied on the case, but rather visualized with a missing step, and the impact of the framework will be presented. The parts of the framework that will be removed after each other

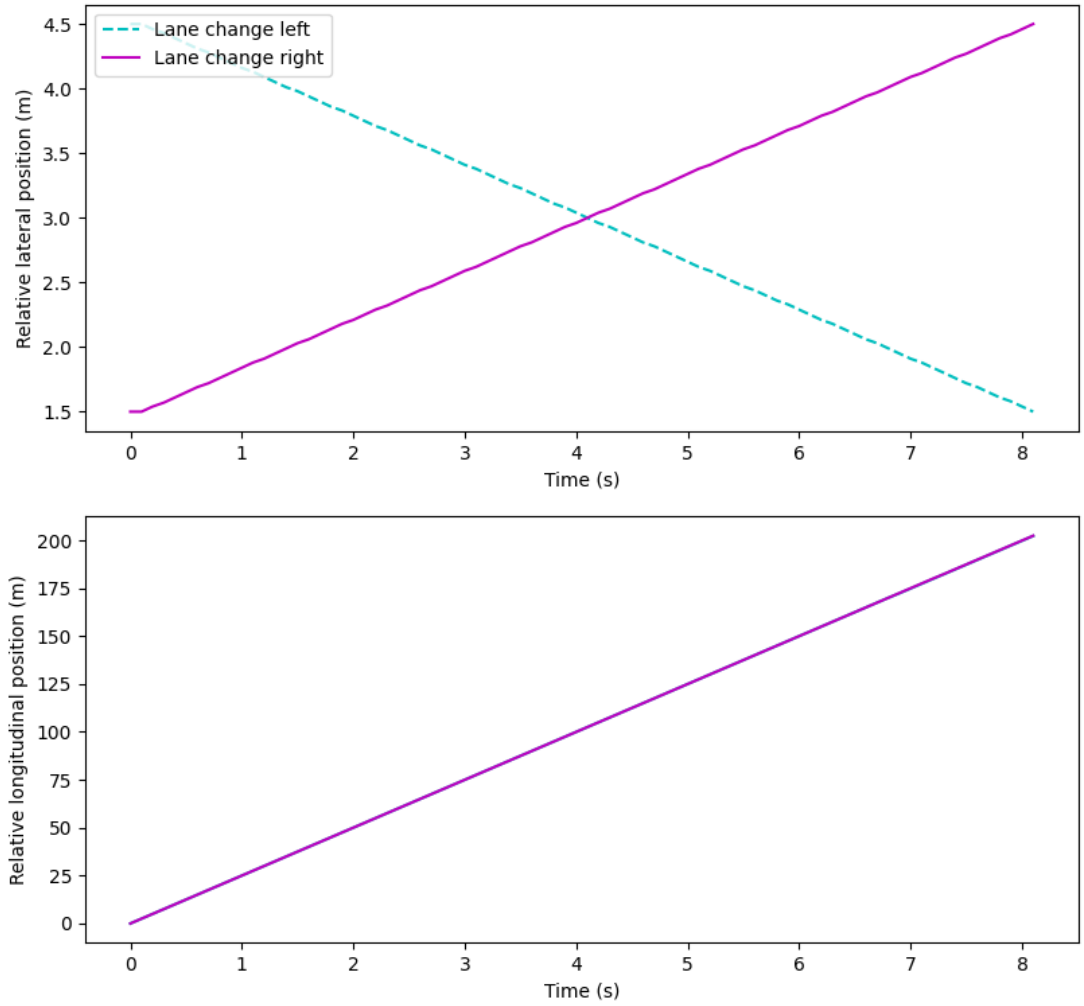


Figure 6.1: Graphs of lateral and longitudinal positions from calculated data in esmini after the injected fault. The upper graph displays relative lateral position (m) over time (s). The bottom graph displays relative longitudinal position (m) over time (s).

are, the assumption document, the automated tests, reconnection with stakeholder, and the empirical/visual validation.

Removing the assumptions document in the example of use, in Section 5.1, would impact the creation of requirements, development of the model and the acceptance tests. Without an assumptions document, there is no foundation to build the requirements upon, since there is no described functionality and the model limitations are unknown. Subsequently, the lack of adequate requirements will translate into a model that is hard to verify and validate, as there are no well-defined definitions to base the model upon. Similarly, it will be difficult to create acceptance tests for the model behavior, as the expected behavior is not defined. In the example of use, the assumptions document specify the limitations and assumptions for the model. An example is the simplifying assumption that all lanes are always seen as switchable if they are drivable. Without this assumption, it will be difficult to validate when a lane change should be performed.

If the automated tests would be removed, it would require a lot more manual work as both the acceptance test and unit tests would not be executed in that case. Instead, a tester would have to manually perform the same steps for all tests by running the model with different start parameters. Additionally, it would be difficult for a tester to ensure that the tests are run exactly the same each time and within a reasonable amount of time. As seen in the injected fault experiment, most of the faults were detected by the automated tests. If these would not be a part of the framework, the faults might not have been detected and could ultimately result in a model that does not pass validation.

Removing the reconnection with stakeholders from the framework would impact the V&V process negatively. As it can be seen in the example of application, in Chapter 5, the development of the model started with three requirements which over the iterations were extended to a total of twelve requirements. Without the reconnection with stakeholders, these additional nine requirements would not be elicited. This would have resulted in that the developed model did not fulfill the expected behavior of stakeholders' desire.

The empirical and visual validation, that can be performed as the last step in the framework, is useful for validating requirements that can not be validated by automated acceptance tests. If this part of the framework was removed, it would result in requirements that can not be validated. An example of this can be seen in the example of use, where the empirical and visual validation is used to validate that requirement 2 and 12 is fulfilled. If removed, these two requirements could not have been validated, resulting in that the developed model could have undesired behavior.

6.2.3 Usefulness and effectiveness

Based upon the evaluation of the framework, it can be seen that the current version of the framework provides usefulness when developing a simulation model in esmini.

By removing parts of the framework, it is seen that they are all essential to the V&V process of esmini, since excluding any of the parts in the framework would result in a less validated model as it would be hard to know if the stated requirements are fulfilled.

The framework proved effective in finding faults in LIRM, due to the many levels of tests as well as visual and empirical validation at the end of each iteration. The effectiveness of the framework on other models is difficult to estimate, as the framework was only applied to a single model developed by the authors.

7
Discussion

This chapter contains the discussion, which will present subjects such as the framework, the application of the framework, future research, threats to the report's validity, and ethics.

7.1 The developed model, LIRM

The lane independent routing model, LIRM, was developed to solve the case from Volvo Cars and the framework was applied to it. LIRM has solved the issue with lane dependent routing that esmini had, by implementing a new controller that extends the behavior of the pathfinding and route following to incorporate lane changes. We felt that the framework provided useful insights and help during the development of LIRM, as it forced us to actually implement requirements and tests, and consider assumptions and limitations before the development began, which is something that is not always performed otherwise. The implementation of LIRM in esmini was performed on a separate Git branch, but will be merged into the main master branch.

7.2 The framework

For the evaluation of the framework, we chose three faults for to inject, seen in Table 6.1. The faults were chosen in such a way that each fault targeted a different requirement and a different step of the framework, in order to show how various parts of the framework can detect a fault in the model. Evidently, there are many other faults that could have been injected to inspect the fault detection of the framework. However, more faults would have taken more time to inject and detect, which ultimately led to the choice of having only three faults as this seemed reasonable to show that the framework detected different kind of faults. We considered the use of mutation testing, as this would have increased number of injected faults and validity of the fault testing. However, we chose not to include it because of time limitations as the visual and empirical validation would have taken a considerable amount of time to perform for each mutation. Though, mutation testing would have been useful for the acceptance and unit tests since these would not require manual work and would have created a comprehensive set of faults.

In feedback interviews with employees at Volvo Cars, the framework seems to have

value in regard to improving the V&V process and consequently the overall development process in esmini. The interviewees stated that our framework would be useful as a starting point for V&V in esmini, and the reason for that might be that they did not apply the framework to gain experience of using it and that they felt unsure about how easily it could be incorporated into the workflow at Volvo Cars. The feedback also included that the framework could be to cumbersome for smaller models or tasks, since the process of stating assumptions and eliciting requirements would be both time consuming and require a lot of manual work in comparison to the size of the model developed. We were aware that this could be one of the drawbacks of the framework, however, we felt that the framework is likely to be more beneficial for the V&V process when focusing on more comprehensive and complex models. For the evaluation of the framework, the practical usefulness and effectiveness of the framework has not yet been fully determined and would require more research to acknowledge.

A major concern with the chosen method was that a proper evaluation study was not included. Consequently, this made it difficult to answer RQ2, evaluating the usefulness and effectiveness, as the evaluation of the framework was primarily based on our own experiences. As mentioned in the Chapter 3, this could have been mitigated with a judgment study conducted as a part of the thesis work. However, we deemed that incorporating such a study likely to exceed a reasonable workload and time frame for a single master thesis. Another approach could have been to focus on applying only a smaller set of the framework to a model. Though, this approach would not enfold the entire framework and its iterative process unless the span of the thesis was only regarding a smaller aspect of the V&V process.

There were several techniques that we found, which were interesting and were possible candidates of techniques to include in the framework. One of these techniques was a weight system described by Raunak and Olsen [17], which is summarized in Section 2.2. This technique could have been helpful to further enhance the V&V process in the framework, as it would have provided a way of calculating the validation coverage of the model. However, we thought that it might not be as useful in our framework due to the lack of statistical validation, as the weight system was mostly used for statistical validation techniques. Another set of techniques that were not explicitly specified in the framework were integration and system tests. The primary reason for not including these, was that they were not emphasized as particularly important by any literature or related work. Neither did we consider them important on their own, as our definition of acceptance tests included the concept of system and integrating testing. They can certainly be incorporated as standalone verification techniques outside the framework, but they were not considered essential for the general V&V process of models in esmini.

The reason for the lack of statistical validation in the framework was that it required large amounts of data in order to be accurate. Since this is something that is not always available for all types of models in esmini, and as esmini is mostly a deterministic simulator, there are not many, if any models that would benefit

from the addition of specific statistical validation techniques. The type of statistical technique to use is highly dependent on the type of data and what it should be validated against, which made it difficult to recommend any specific techniques in general. Our case and developed model did not have real data that could be used statistically nor did the model have requirements that needed statistical validation, therefore the use of statistical validation was not deemed necessary nor beneficial to include in the framework.

The framework is not meant to provide exhaustive testing of the model it is applied on, as this could be considered unreachable. However, it is designed to allow the stakeholders and developers to reach a decided degree of coverage for the V&V process, as reaching a 100% coverage degree can be considered impossible. This is one of the reasons for using CQAS to create acceptance tests, as CQAS allows for creating scenarios that test important parts of the model to reach the desired coverage degree.

A benefit that the framework can provide to the development of models is to help with the modification or correction of already existing models, to either accommodate for new requirements or calibrate faulty models.

7.3 Uniqueness of framework

The developed framework differentiates itself from most other found frameworks, presented in Section 2.2, as it concretely defines the process needed to ensure that V&V is performed on the developed model by presenting a set of steps that need to be performed during the entire development process of the framework. This differs from the other frameworks or concepts that focuses more on stating many different techniques that could be useful depending on the model, such as Roungas et al.'s [12] table of suggested V&V methods. We did not find any other frameworks that included more extensive ideas and concepts from software engineering, such as ATDD and CQAS, to better enhance the V&V process of simulation models. Another adaptation that makes the framework unique, is the complete lack of statistical validation techniques, as these validation techniques were the most common in the other frameworks. This was, as stated earlier, not implemented as statistical validation has little to no value for simulation models in esmini.

The motivation for including the assumptions document in the framework is that esmini does not directly adapt reality but has the ASAM standards as a layer between reality and the simulation, which causes more assumptions and limitations than a direct adaptation of reality which many other simulators are. We chose to include ATDD, CQAS and user stories to improve the understandability of requirements and the connection between tests and requirements, as this connection can otherwise be difficult to understand. This choice was not specific to esmini, but rather to improve the framework in general by reducing the risk of miscommunication and misunderstanding between stakeholders and developers. The inclusion of our definition of refactoring and calibration should not be seen as a standalone

V&V technique, but rather included in the framework to make sure that models are improved in non-functional aspects. The idea of reconnecting with stakeholders came from ATDD and agile practices, and was further motivated by the fact that many stakeholders for esmini work at Volvo Cars which makes it possible to perform this step without losing too much time. The main benefit of reconnecting with stakeholders is to make sure that the right model is being created, therefore increasing both validity and credibility. The last step in the framework is to perform visual and empirical validation. Visual validation was suitable for esmini as most models can be performed by running and visually inspect the result, which is commonly easier than comparing data to validate the model. This is further motivated by most esmini models being deterministic and results not changing between runs. Empirical validation is necessary to help validate sub-components within models that require data analysis to be determined as accurate, such as the lane change modeling performed in the developed model, LIRM.

7.4 Threats to validity

Threats to internal validity: The framework could make faulty assessments due to selection bias during the creation of the framework. This could possibly affect the effectiveness of the framework and the simulation model it was applied on. Certain V&V methods included in the framework might be there as a result of the bias of the researchers, for example, opinions could interfere. Consequently, this can have led to an inefficient method usage.

Another threat to internal validity could be that the same persons that developed the framework are the ones that performed the experiment, where the framework is applied to a model. The developers of the framework are also the ones that evaluate it. This could have led to a biased usefulness result of the framework, as the people performing the experiment will already have in-depth knowledge of the framework compared to an outsider. This could have been mitigated by performing a study where other developers used or evaluated the framework, however, as mentioned before this was not feasible due to the time limit. Instead, we tried to mitigate this form of bias by receiving feedback and evaluation from developers and users at Volvo Cars.

Threats to external validity: There is a risk that the simulation environment that the framework was applied on does not represent a general environment, which could have led to the framework being non-generalizable. The framework was only applied to a singular class of simulation models, which might also have an impact on the generalizability and usefulness of other models. Applying the framework to multiple model classes to mitigate this threat was deemed infeasible due to the time limit of the thesis.

7.5 Future research

The presented framework should not be interpreted as the last and final version for validation and verification in regard to esmini, on the contrary there are several subjects that could be further researched to improve the frameworks usability and effectiveness. Automation is one of these subjects that the framework would benefit greatly on, as most steps and task currently are time-consuming to do, as they are almost entirely manual. If more research went into automation of tests and test generation for models in esmini, the usability of the framework would increase greatly.

In the framework's current shape, all steps are performed manually, except for the execution of tests. Section 2.8 describes a couple of testing techniques that assist in automating other testing activities. Two mentioned testing approaches that promote automated test generation are MBT and PBT. Both of these could most likely be incorporated into the framework in one way or another. Though, manual work is still required for both of them in an earlier step. For PBT, the properties would have to be manually devised. For MBT, the modeled UML diagram(s) of the system need to be manually traced according to the stated requirements. Presumably, much time is spent on developing test cases and either one of these techniques contribute to less effort spent on test case design.

Another area of testing to explore for future work is proposed by Gambi et al. [42], see Section 2.8. The proposed idea is to procedurally generate road networks used to test virtual self-driving vehicles in challenging virtual environments. A similar approach could be utilized for esmini and our framework, where various types of road networks are generated to test vehicle behavior or controller functionality. The automatic generation of road networks puts the focus on expected behavior and requirements, in contrast to putting a great effort on the creation of valid road networks.

Another subject that could benefit from more research is the generalizability of the framework, since as of now it has only been deployed on esmini models, but we see that the framework has potential to be applied in other simulators. As the main ideas that the framework consists of are not specific for esmini, the assumptions document and the requirements with CQAS and acceptance tests are general concepts that should be directly applicable to other simulators. The iterative steps of reconnecting with stakeholders and the visual and empirical validation could however need to be adapted to be applicable to other simulators. This could be supported with more studies on the current framework on other models and simulators than the example provided in the thesis, as studies on other models could identify problems with the presented framework and further increase the effectiveness of the framework.

To improve the efficiency and usability of the framework, another idea for future work could be to develop a workflow management program for the framework. The program could help the user with the creation of assumption documents, require-

ments and CQAS to reduce the time spent on manually writing the documents for these part. Additionally, such a program could be beneficial in introducing new users to the framework.

7.6 Ethics

Since esmini is a traffic simulator with the functionality to be extended to test software for actual vehicles, such as safety functions or self-driving functionality, it is important to keep the models that are developed for esmini realistic so that they do correspond to reality at the wanted degree. One step to ensure this is through validation and verification of the models, which this framework aims to provide a foundation for. If the models are not validated, the tested software might behave differently in the simulation to what it will in reality, which possibly could have a real world impact as it could result in accidents. Therefore, from a safety critical standpoint, the framework is important to help ensure that the models in esmini are validated and verified for their use cases.

If concrete statistical validation techniques were to be implemented into the framework, another ethical issue arises in the form of data management. For statistical validation to be performed, large amounts of data is required and this creates its own problems such as how the data is collected, who should have access to it, how can data privacy be ensured, and many more. Therefore, this is an important issue to contemplate when considering statistical validation, which is one of the reasons why statistical validation was not prioritized in the framework.

8

Conclusion

This chapter will contain the conclusion of the thesis. The conclusion will provide the authors' input on how successful and useful the framework is, as well as summarizing the entire thesis.

The aim of this thesis was to create a V&V framework for esmini and evaluate it based on usefulness and effectiveness. Before creating the framework, time was spent on researching existing V&V methods for simulation models as well as general V&V techniques used in software engineering. Subsequently, the developed framework was created by combining found methodologies and ideas from the authors into an iterative procedure that encapsulates the entire development process of a model. The framework was applied during the development of a simulation model to provide an example of application to ease the understanding of the framework. The developed simulation model, LIRM, has solved the issues esmini had in regard to pathfinding. The evaluation of the framework was performed by fault injections, removal of parts of the framework and feedback from developers in esmini. Furthermore, there are still aspects of the framework that could be improved, such as automating the test creation process to allow for less time to be spent on manual implementation of tests. From the results and feedback collected during the application and evaluation of the framework, we conclude that the framework provide helpful guidance to the V&V process and is an important stepping stone in the research of improving V&V for simulation models in esmini. The framework has also proven that it could be useful for other similar simulators, as our framework provides guidance for including V&V into an iterative development and calibration process. Additionally, we believe that the application example of the framework supports the conclusion that the framework improves the V&V process. One of the main contributions to research is the concreteness of the developed framework, in comparison to existing frameworks, with a clear visualization and explicit steps to apply. Another contribution to research is the addition of a complete V&V process for esmini, as no such process currently exists, that is also possibly useful for other simulators with minor modifications.

References

[1] IEEE, "IEEE Standard for System, Software, and Hardware Verification and Validation," in *IEEE Std 1012-2016 (Revision of IEEE Std 1012-2012/ Incorporates IEEE Std 1012-2016/Cor1-2017)*, 2017, pp. 1–260. DOI: 10.1109/IEEESTD.2017.8055462.

[2] MITRE, *System engineering guide: Verification and validation of simulation models*, 2014. [Online]. Available: https://www.mitre.org/publications/systems-engineering-guide/se-lifecycle-building-blocks/other-se-lifecycle-building-blocks-articles/verification-and-validation-of-simulation-models (visited on 2021-12-10).

[3] O. Karlsson and E. Fredin, "Development of a controller to switch between relative and absolute path for target vehicles in simulation scenarios," *Department of Automatic Control, Lund University, Sweden*, 2021. [Online]. Available: https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=9061680&fileOId=9061681.

[4] C. Chau and Q. Liu, "Driver modelling for virtual safety assessment of automated vehicle functionality in cut-in scenarios," *Department of Mechanics and Maritime Sciences, Chalmers University of Technology, Göteborg, Sweden*, 2021. [Online]. Available: https://odr.chalmers.se/bitstream/20.500.12380/304058/1/2021-55%5C%20Christoffer%5C%20Chau%5C%20%5C%26%5C%20Qianyu%5C%20Liu.pdf.

[5] C. Beisbart and N. J. Saam, *Computer Simulation Validation: Fundamental Concepts, Methodological, and Philosophical Perspectives*. Switzerland: Springer Nature Switzerland AG, 2019.

[6] R. G. Sargent, "Verification and validation of simulation models," in *Proceedings of the 2011 Winter Simulation Conference*, Baltimore, USA, 2010, pp. 166–183. DOI: 10.1109/WSC.2010.5679166.

[7] S. Riedmaier, B. Danquah, B. Schick, and F. Diermeyer, "Unified framework and survey for model verification, validation and uncertainty quantification," *Archives of Computational Methods in Engineering*, vol. 28, pp. 2655–2688, 2021. DOI: 10.1007/s11831-020-09473-7.

[8] Euro NCAP, *Euro NCAP protocols*, 2022. [Online]. Available: https://www.euroncap.com/en/for-engineers/protocols/ (visited on 2022-05-13).

[9] ASAM, *ASAM OpenSCENARIO*, 2021. [Online]. Available: https://www.asam.net/standards/detail/openscenario/ (visited on 2021-12-06).

[10] VTI, *Driving Simulation*, 2020. [Online]. Available: https://www.vti.se/en/research/vehicle-technology-and-driving-simulation/driving-simulation (visited on 2021-12-13).

[11] ISTQB, *Certified Tester Foundation Level (CTFL) Syllabus*, Version 2018 v3.1.1, 2021. [Online]. Available: https://istqb-main-web-prod.s3.amazonaws.com/media/documents/ISTQB-CTFL_Syllabus_2018_v3.1.1.pdf (visited on 2022-05-05).

[12] B. Roungas, S. Meijer, and A. Verbraeck, "A framework for optimizing simulation model validation & verification," *International Journal on Advances in Systems and Measurements*, vol. 11, pp. 137–152, 2018. [Online]. Available: http://resolver.tudelft.nl/uuid:53a9fbfe-5d9a-40df-924d-94ee49bd21ae.

[13] J. Barceló, "Models, traffic models, simulation, and traffic simulation," in *Fundamentals of Traffic Simulation*, J. Barceló, Ed. New York, NY: Springer New York, 2010, pp. 1–62, ISBN: 978-1-4419-6142-6. DOI: 10.1007/978-1-4419-6142-6_1. [Online]. Available: https://doi.org/10.1007/978-1-4419-6142-6_1.

[14] J. Barceló, "Models, traffic models, simulation, and traffic simulation," in *Fundamentals of Traffic Simulation*, J. Barceló, Ed. New York, NY: Springer New York, 2010, pp. 269–293, ISBN: 978-1-4419-6142-6. DOI: 10.1007/978-1-4419-6142-6_1. [Online]. Available: https://doi.org/10.1007/978-1-4419-6142-6_1.

[15] A. M. Law, "How to Build Valid and Credible Simulation Models," in *2019 Winter Simulation Conference (WSC)*, 2019, pp. 1402–1414. DOI: 10.1109/WSC40007.2019.9004789.

[16] N. David, N. Fachada, and A. C. Rosa, "Verifying and validating simulations," in *Simulating Social Complexity: A Handbook*, B. Edmonds and R. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 173–204, ISBN: 978-3-319-66948-9. DOI: 10.1007/978-3-319-66948-9_9. [Online]. Available: https://doi.org/10.1007/978-3-319-66948-9_9.

[17] M. Raunak and M. Olsen, "Quantifying validation of discrete event simulation models," in *Proceedings of the Winter Simulation Conference 2014*, 2014, pp. 628–639. DOI: 10.1109/WSC.2014.7019927.

[18] Z. Wang and A. Lehmann, "A Framework for Verification and Validation of Simulation Models and Applications," in *AsiaSim 2007*, J.-W. Park, T. G. Kim, and Y.-B. Kim, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 237–246, ISBN: 978-3-540-77600-0.

[19] C. Yin and A. McKay, "Introduction to modeling and simulation techniques," in *Proceedings of ISCIIA 2018 and ITCA 2018. The 8th International Symposium on Computational Intelligence and Industrial Applications and The 12th China-Japan International Workshop on Information Technology and Control Applications*, Tengzhou, China, 2018. [Online]. Available: https://www.researchgate.net/publication/332962311_Introduction_to_Modeling_and_Simulation_Techniques (visited on 2022-05-09).

[20] N. Azlan and M. Rohani, "Overview Of Application Of Traffic Simulation Model," *MATEC Web of Conferences*, vol. 150, pp. 1–51, 2018. DOI: https://doi.org/10.1051/matecconf/201815003006.

[21] A. Bazghandi, "Techniques, Advantages and Problems of Agent Based Modeling for Traffic Simulation," *IJCSI International Journal of Computer Science Issues*, vol. 9, pp. 115–119, 2012, ISSN: 1694-0814. [Online]. Available: https://www.ijcsi.org/papers/IJCSI-9-1-3-115-119.pdf.

[22] J. Nguyen, S. T. Powers, N. Urquhart, T. Farrenkopf, and M. Guckert, "An overview of agent-based traffic simulators," *Transportation Research Interdisciplinary Perspectives*, vol. 12, 2021, ISSN: 2590-1982. DOI: https://doi.org/10.1016/j.trip.2021.100486.

[23] ASAM, *ASAM, About ASAM*, 2021. [Online]. Available: https://www.asam.net/about-asam/our-vision/ (visited on 2022-02-09).

[24] ASAM, *The History of ASAM*, 2021. [Online]. Available: https://www.asam.net/about-asam/history/ (visited on 2022-02-09).

[25] ASAM, *Technology*, 2021. [Online]. Available: https://www.asam.net/about-asam/technology/ (visited on 2022-02-09).

[26] ASAM, *ASAM OpenDRIVE*, 2021. [Online]. Available: https://www.asam.net/standards/detail/opendrive/ (visited on 2021-12-06).

[27] ASAM, *ASAM OpenDRIVE, Version: 1.7.0*, 2021. [Online]. Available: https://www.asam.net/index.php?eID=dumpFile&t=f&f=4422&token=e590561f3c39aa2260e5442e29e93f6693d1cccd#top-e8fe3504-e54c-48b9-bddc-99c4fa67631e (visited on 2022-02-09).

[28] ASAM, *ASAM OpenSCENARIO: User Guide, Version: 1.1.1*, 2021. [Online]. Available: https://www.asam.net/index.php?eID=dumpFile&t=f&f=4596&

token=55bca7d8439f2bae072c4dffd1ee544a6d76b786 (visited on 2022-02-09).

[29] E. Knabe, *Simulation scenarios*, 2019. [Online]. Available: https://sites.google.com/view/simulationscenarios (visited on 2021-12-06).

[30] esmini, *esmini*, 2021. [Online]. Available: https://github.com/esmini/esmini (visited on 2021-12-06).

[31] esmini, *Inner Workings of esmini*, 2021. [Online]. Available: https://github.com/esmini/esmini/blob/master/docs/InnerWorkings.md (visited on 2022-03-02).

[32] E. Knabe, *Controllers in esmini*, 2020. [Online]. Available: https://github.com/esmini/esmini/blob/master/docs/Controllers.md (visited on 2022-02-11).

[33] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[34] P. A. Lopez, M. Behrisch, L. Bieker-Walz, *et al.*, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*, IEEE, 2018. [Online]. Available: https://elib.dlr.de/127994/.

[35] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. USA: Pearson Education, Inc, 2015, pp. 63–69.

[36] "Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models," International Organization for Standardization, Standard ISO/IEC 25010:2011, Mar. 2011.

[37] L. Koskela, *Test Driven: Practical TDD and Acceptance TDD for Java Developers*, ser. Manning Pubs Co Series. Manning, 2008, ch. 9, pp. 323–362, ISBN: 9781932394856.

[38] K. Pugh, *Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration*, 1st ed., ser. Net Objectives Lean-Agile Series. Addison-Wesley Professional, 2010, ISBN: 9780321714084.

[39] V. Garousi and F. Elberzhager, "Test automation: Not just for test execution," *IEEE Software*, vol. 34, no. 2, pp. 90–96, 2017. DOI: 10.1109/MS.2017.34.

[40] I. Schieferdecker, "Model-based testing," *IEEE Software*, vol. 29, pp. 14–18, Jan. 2012. DOI: 10.1109/MS.2012.13.

[41] A. Löscher and K. Sagonas, "Targeted property-based testing," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2017, Santa Barbara, CA, USA: Association for Computing Machinery, 2017, pp. 46–56, ISBN: 9781450350761. DOI: 10.1145/3092703.3092711. [Online]. Available: https://doi.org/10.1145/3092703.3092711.

[42] A. Gambi, M. Mueller, and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation," Jul. 2019, pp. 318–328. DOI: 10.1145/3293882.3330566.

[43] K. Peffers, T. Tuunanen, and M. A. Rothenberger, "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007. DOI: 10.2753/MIS0742-1222240302.

[44] T. J. Ellis and Y. Levy, "A Guide for Novice Researchers: Design and Development Research Methods," in *Proceedings of Informing Science & IT Education Conference (InSITE) 2010*, vol. 10, 2010, pp. 107–118. [Online]. Available: http://proceedings.informingscience.org/InSITE2010/InSITE10p107-118Ellis725.pdf (visited on 2021-12-14).

[45] S. Eismann, J. Walter, J. von Kistowski, and S. Kounev, "Modeling of Parametric Dependencies for Performance Prediction of Component-Based Software Systems at Run-Time," in *2018 IEEE International Conference on Software Architecture (ICSA)*, 2018, pp. 135–139. DOI: 10.1109/ICSA.2018.00023.

[46] K.-J. Stol and B. Fitzgerald, "The ABC of Software Engineering Research," *ACM Transactions on Software Engineering and Methodology*, vol. 27, no. 3, pp. 1–51, 2018. DOI: https://doi.org/10.1145/3241743.

[47] A. Candra, M. A. Budiman, and K. Hartanto, "Dijkstra's and A-Star in Finding the Shortest Path: a Tutorial," in *2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)*, 2020, pp. 28–32. DOI: 10.1109/DATABIA50434.2020.9190342.

A

Lane Independent Router Case from Volvo

Case: Lane independent routing with lane changes

When simulating a road network, and a vehicle in this network, a route can be defined to determine how a vehicle should move in e.g., an intersection to follow this route. In the present esmini implementation of this, this can only be done if the waypoints in the route are connected to the lane the vehicle is presently in. This thesis should make a lane independent route strategy, where a vehicle can change lanes in order to take the correct path for a route.

Objectives

- Vehicles can find their way through a road network without having to be placed in the correct lane
- A route following controller has to be created, implementing the routing and modeling of lane changes.
- The model should handle lane changing with some degree of realism, for example, maybe not change lane 2 meters before an exit on a highway.
- The routing should support three different routing strategies as defined in OpenSCENARIO, shortest, fastest and least number of intersections.

Assumptions:

- Realism is not the highest priority, but could be added with the use of credible parameters
- Road networks are not infinite / have some sort of limit on the size
- Lane can't connect with lane with different heading (no U-turn)
- The target lane must be defined as drivable for the algorithm to find it (there are different kinds of lanes in OpenDRIVE, e.g., borders, sidewalks, driving, biking, etc.)
- All roads or lanes that the vehicle passes through must also be defined as drivable.
- All lanes are always seen as switchable
    - On a specific road, assume that the vehicle can always change lane to another lane in the same direction as it is driving

Example:

Looking at the picture below of a road network in esmini (see Figure 1). The current implementation of routing allows the yellow vehicle to reach the red dot, and the green vehicle to reach the pink dot. The goal of the lane independent routing with lane changes is that both vehicles should be able to reach both dots. As well as actually changing lanes when driving.

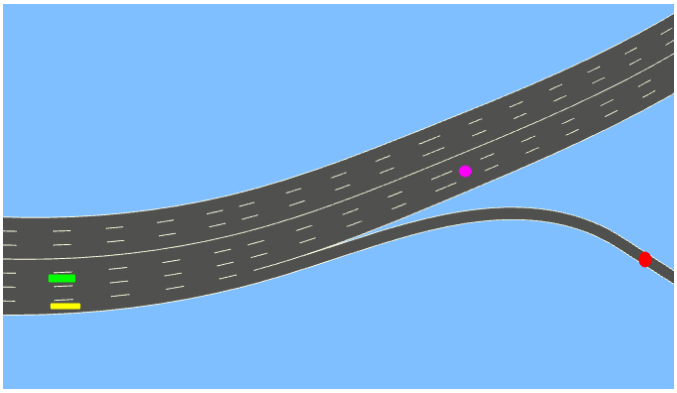


Figure 1

B

All Requirements and CQASs

Requirements:

The requirements for the lane independent routing model.

Lane independent path finding (LIPF):

User Story	*As a <role>, I want <functionality> because <reason>.*
1	As a developer, I want the algorithm to calculate a path within a reasonable amount of time for the given size of OpenDRIVE road network, as it should not slow down the rest of the simulation.
2	As a developer, I want the algorithm to find the expected/optimal path for a given route strategy, since it should be close to an ideal pathfinder.
4	As a developer, I want the algorithm to be able to find a path (sequence of roads to traverse) between the vehicle position to a valid target, so that the simulated vehicle can reach the target.
5	As a stakeholder, I want the algorithm to find the path based on the routing strategy “shortest”, as it is defined in the OpenSCENARIO specification.
6	As a stakeholder, I want the algorithm to find the path based on the routing strategy “fastest”, as it is defined in the OpenSCENARIO specification.
7	As a stakeholder, I want the algorithm to find the path based on the routing strategy “the least number of intersections”, as it is defined in the OpenSCENARIO specification.
8	As a developer, I don’t want esmini to crash if the algorithm can’t find a valid path, as I want to be informed of this instead and keep esmini running.
9	As a developer, I want the functionality to translate paths to waypoints, so that the model can follow them.

Lane change controller (LCC):

User Story	*As a <role>, I want <functionality> because <reason>.*
3	As a developer, I want the modeled vehicle to be able to reach the target waypoint, in order to follow the path created by the lane independent pathfinder.
10	As a user, I want the modeled vehicle to be able to change lane, in order to follow the path created by the lane independent pathfinder.
11	As a tester, I don't want the lane change to happen if it results in a collision, as it would be “unrealistic”.
12	As a tester, I want the lane change to have a degree of realism, since the model could be used for vehicle testing.

CQAS:

The concrete quality attribute scenarios for each requirement.

Requirement	LIPF - User Story 1.
Quality Attribute	Performance
Stimulus	A valid target position
Source of stimulus	User or read from the OpenSCENARIO file
Response	A path, (sequence of roads to traverse)
Response measure	How long it takes to find / calculate the path
Artifact	The lane independent path finding algorithm
Environment	Runtime, after a new target as been received

Requirement	LIPF - User Story 2.
Quality Attribute	Correctness
Stimulus	A valid target
Source of stimulus	User or read from the OpenSCENARIO file
Response	The correct/optimal path, (sequence of roads to traverse)
Response measure	The found path
Artifact	The lane independent path finding algorithm
Environment	Runtime, after a new target as been received

Requirement	LCC - User Story 3.
Quality Attribute	Functional completeness / Correctness
Stimulus	A list of waypoints
Source of stimulus	After path has been found, and waypoints has been created
Response	The modeled vehicle drives to target.
Response measure	If the modeled vehicle has reached target
Artifact	Lane change controller
Environment	Runtime, after waypoints has been created

Requirement	LIPF - User Story 4.
Quality Attribute	Correctness
Stimulus	A valid target position
Source of stimulus	User or read from the OpenSCENARIO file
Response	A path, (sequence of roads to traverse)
Response measure	If it finds a path
Artifact	The lane independent path finding algorithm
Environment	Runtime, after a new target as been received

Requirement	LIPF - User Story 5.
Quality Attribute	Functional completeness
Stimulus	A valid target position, with route strategy: shortest
Source of stimulus	User or read from the OpenSCENARIO file
Response	A path, (sequence of roads to traverse)
Response measure	If it finds a path
Artifact	The lane independent path finding algorithm
Environment	Runtime, after a new target as been received

Requirement	LIPF - User Story 6.
Quality Attribute	Functional completeness
Stimulus	A valid target position, with route strategy: fastest
Source of stimulus	User or read from the OpenSCENARIO file
Response	A path, (sequence of roads to traverse)
Response measure	If it finds a path
Artifact	The lane independent path finding algorithm
Environment	Runtime, after a new target as been received

Requirement	LIPF - User Story 7.
Quality Attribute	Functional completeness
Stimulus	A valid target position, with route strategy: least number of intersections
Source of stimulus	User or read from the OpenSCENARIO file
Response	A path, (sequence of roads to traverse)
Response measure	If it finds a path
Artifact	The lane independent path finding algorithm
Environment	Runtime, after a new target as been received

Requirement	LIPF - User Story 8.
Quality Attribute	Fault tolerance
Stimulus	An invalid target position or no path found
Source of stimulus	User or read from the OpenSCENARIO file
Response	A path, (sequence of roads to traverse)
Response measure	If the application crashes or not.
Artifact	The lane independent path finding algorithm
Environment	Runtime, after a new target as been received

Requirement	LIPF - User Story 9.
Quality Attribute	Functional completeness / Correctness
Stimulus	A path of nodes
Source of stimulus	After path has been found, return value from pathfinder
Response	A list of waypoints (path to target)
Response measure	If a list has been created
Artifact	The lane independent routing - waypoint creator
Environment	Runtime, after a path has been found

Requirement	LCC - User Story 10.
Quality Attribute	Functional completeness / Correctness
Stimulus	A waypoint in another lane
Source of stimulus	After path has been found, and waypoints has been created
Response	The model vehicle change lane to the one specified in waypoint
Response measure	If the model vehicle has changed lane
Artifact	Lane change controller
Environment	Runtime, after a list of waypoints has been acquired

Requirement	LCC - User Story 11.
Quality Attribute	Functional completeness / Correctness
Stimulus	A lane change
Source of stimulus	A waypoint in another lane
Response	Change lane if no vehicle in adjacent lane, otherwise recalculate path to target
Response measure	If lane change occurs and no collision
Artifact	Lane change controller
Environment	Runtime, when lane change controller is called

Requirement	LCC - User Story 12.
Quality Attribute	Functional completeness / Correctness
Stimulus	A lane change
Source of stimulus	A waypoint in another lane
Response	Change lane if distance to end of road is more than specified threshold
Response measure	If lane change occurs, how it looks
Artifact	Lane change controller
Environment	Runtime, when lane change controller is called

Acceptance tests:

Small OpenDRIVE road network = highway_example_with_merge_and_split.xodr
Medium OpenDRIVE road network = multi_intersections.xodr
Large OpenDRIVE road network = large_network.xodr
In road direction: Heading = 0 radians
Opposite of road direction: Heading = π radians (relative to road direction)

Requirement: LIPF - User Story 1.

OpenDRIVE: Medium
Type: Code

Start position: s=100, t=0, road id = 202, lane id = 2
Target position: s=20, t=0, road id = 209, lane id = 1
Driving direction: opposite of road
startTime = save current time
path = result from pathfinder function
endTime = save current time
Verify that the returned path list is not empty
Verify that the last node in path has road id 209
calcTime = endTime - startTime
Verify that calcTime is less than < 5ms

OpenDRIVE: Large
Type: Code

Start position: s=1100, t=0, road id = 2291, lane id = -1
Target position: s=50, t=0, road id = 2502, lane id = 1
Driving direction: in road
startTime = save current time
path = result from pathfinder function
endTime = save current time
Verify that the returned path list is not empty
Verify that the last node in path has road id 2502
calcTime = endTime - startTime
Verify that calcTime is less than < 15ms

Requirement: LIPF - User Story 2.

OpenDRIVE: Medium
Type: Visual

1. Trace the OpenDRIVE road network and calculate the expected route based on each route strategy from a start position to a target position for 5 routes. This step could be automated in the future, but was performed manually in this case due to time constraints.
2. Enable path logger in the simulation model.
3. Run the simulation model, with the start and target positions of the 5 routes.
   Collect the calculated route for each route from the model.
4. Use the expected route and the collected calculate route
   Visually inspect the routes and compare them to determine if the calculated path takes the same route as the expected.

Requirement: LIPF - User Story 2&5.

OpenDRIVE: Medium
Type: Code
Start position: s=100, t=0, road id = 202, lane id = 2
Target position: s=20, t=0, road id = 209, lane id = 1
Route strategy = shortest
Driving direction: opposite of road
path = result from pathfinder function
Verify that the returned path list is not empty
Verify that the last node in path has road id 209

Requirement: LIPF - User Story 2&6.

OpenDRIVE: Medium
Type: Code
Start position: s=100, t=0, road id = 202, lane id = 2
Target position: s=20, t=0, road id = 209, lane id = 1
Route strategy = fastest
Driving direction: opposite of road
path = result from pathfinder function
Verify that the returned path list is not empty
Verify that the last node in path has road id 209

Requirement: LIPF - User Story 2&7.

OpenDRIVE: Medium
Type: Code
Start position: s=100, t=0, road id = 202, lane id = 2
Target position: s=20, t=0, road id = 209, lane id = 1
Route strategy = least number of intersections
Driving direction: opposite of road
path = result from pathfinder function
Verify that the returned path list is not empty
Verify that the last node in path has road id 209

Requirement: LCC- User Story 3.

OpenDRIVE: Small
Type: Code
Start position: s=10, t=0, road id = 0, lane id = -1
Target position: s=20, t=0, road id = 2, lane id = -1
Driving direction: in road
run followRouteController with the start and target positions
Verify that vehicle has reached target

Requirement: LIPF - User Story 4.

OpenDRIVE: Small
Type: Code
Start position: s=10, t=0, road id = 0, lane id = -1
Target position: s=20, t=0, road id = 5, lane id = -2
Driving direction: in road
path = result from pathfinder function
Verify that the returned path list is not empty
Verify that the last node in path has road id 5

OpenDRIVE: Small
Type: Code
Start position: s=10, t=0, road id = 0, lane id = -1
Target position: s=20, t=0, road id = 2, lane id = -1
Driving direction: in road
path = result from pathfinder function
Verify that the returned path list is not empty
Verify that the last node in path has road id 2

OpenDRIVE: Medium
Type: Code
Start position: s=100, t=0, road id = 202, lane id = 2
Target position: s=20, t=0, road id = 209, lane id = 1
Driving direction: opposite of road
path = result from pathfinder function
Verify that the returned path list is not empty
Verify that the last node in path has road id 209

OpenDRIVE: Medium
Type: Code
Start position: s=50, t=0, road id = 217, lane id = -1
Target position: s=50, t=0, road id = 275, lane id = -1
Driving direction: in road
path = result from pathfinder function
Verify that the returned path list is not empty
Verify that the last node in path has road id 275

OpenDRIVE: Large
Type: Code
Start position: s=50, t=0, road id = 5147, lane id = -1
Target position: s=100, t=0, road id = 2206, lane id = 1
Driving direction: in road
path = result from pathfinder function
Verify that the returned path list is not empty
Verify that the last node in path has road id 2206

OpenDRIVE: Large
Type: Code
Start position: s=1900, t=0, road id = 1006, lane id = 3
Target position: s=20, t=0, road id = 2203, lane id = 1
Driving direction: opposite of road
path = result from pathfinder function
Verify that the returned path list is not empty
Verify that the last node in path has road id 2203

OpenDRIVE: Large
Type: Code
Start position: s=200, t=0, road id = 8277, lane id = 1
Target position: s=50, t=0, road id = 2291, lane id = -2
Driving direction: opposite of road
path = result from pathfinder function
Verify that the returned path list is not empty
Verify that the last node in path has road id 2291

OpenDRIVE: Large
Type: Code
Start position: s=2200, t=0, road id = 2431, lane id = 1
Target position: s=3700, t=0, road id = 2503, lane id = -1
Driving direction: opposite of road
path = result from pathfinder function
Verify that the returned path list is not empty
Verify that the last node in path has road id 2503

Requirement: LIPF - User Story 8.

OpenDRIVE: Medium
Type: Code
Start position: s=100, t=0, road id = 202, lane id = 2
Target position: s=20, t=0, road id = 1000, lane id = 1
Driving direction: opposite of road
path = result from pathfinder function
Verify that function logs “path was not found”
Verify that no exception happened

OpenDRIVE: Medium
Type: Code
Start position: s=100, t=0, road id = 202, lane id = 2
Target position: s=20, t=0, road id = 209, lane id = 3
Driving direction: opposite of road
path = result from pathfinder function
Verify that function logs “path was not found”
Verify that no exception happened

Requirement: LIPF - User Story 9.

OpenDRIVE: Small
Type: Code
- Start position: s=10, t=0, road id = 0, lane id = -1
- Target position: s=20, t=0, road id = 2, lane id = -1
- Driving direction: in road
- path = result from pathfinder function
- waypoints = waypointCreator(path)
- Verify that waypoint not null
- Position(road,laneid,sPos,Offset)
- expected waypoints = ((0,-4,125,0),(4,-1,50,0),(2,-1,25,0))
- Verify that calculated waypoints is the same as expected

OpenDRIVE: Small
Type: Code
- Start position: s=10, t=0, road id = 0, lane id = -3
- Target position: s=20, t=0, road id = 5, lane id = -3
- Driving direction: opposite road
- path = result from pathfinder function
- waypoints = waypointCreator(path)
- Verify that waypoint not null
- Position(road,laneid,sPos,Offset)
- expected waypoints = ((0,-3,100,0),(3,-3,25,0),(1,-3,57.5,0),(6,-3,15,0),(5,-3,20,0))
- Verify that calculated waypoints is the same as expected

OpenDRIVE: Medium
Type: Code
- Start position: s=50, t=0, road id = 266, lane id = 1
- Target position: s=55, t=0, road id = 275, lane id = 1
- Driving direction:opposite road
- path = result from pathfinder function
- waypoints = waypointCreator(path)
- Verify that waypoint not null
- Position(road,laneid,sPos,Offset)
- expected waypoints = ((266,1,54.5,0),(258,-1,8.85,0),(261,-154.5,0),
(196,1,54,5,0),(204,-1,11.5,0),(197,-1,54,0),(275,1,55,0))
- Verify that calculated waypoints is the same as expected

OpenDRIVE: Large
Type: Code

Start position: s=116, t=0, road id =5219, lane id = -1
Target position: s=17.5, t=0, road id = 2203, lane id = -1
Driving direction: in road
path = result from pathfinder function
waypoints = waypointCreator(path)
Verify that waypoint not null
Position(road,laneid,sPos,Offset)
expected waypoints = ((5219,-2,226.8,0),(8264,-2,11,0),(2357,-1,150.6,0),
(2505,-1,8.7,0),(2704,-1,21.8,0),(2701,-1,8.7,0),(2512,-1,765.5,0),(2203,-1,17.5,0))
Verify that calculated waypoints is the same as expected

Requirement: LCC - User Story 10.

OpenDRIVE: Small
Type: Code

Start position: s=10, t=0, road id = 0, lane id = -1
Target position: s=20, t=0, road id = 3, lane id = -2
Driving direction: in road
run followRouteController with the start and target positions
Verify that vehicle is in target lane

Requirement: LCC - User Story 11.

OpenDRIVE: Small
Type: visual

1. Create scenario with two vehicles, one with a “dumb” behavior and one with the FollowRouteController enabled
2. Have the vehicles drive next to each other
3. Make the vehicle with the controller enabled try to change lane
    a. Verify that it does not change lane if the dumb vehicle is in the way

Requirement: LCC - User Story 12.

OpenDRIVE: Small
Type: Empirical

1. Collect the calculated data from the model during lane changes. Record the lateral change and the longitudinal change over time.
2. Compare the calculated data with the existing expected data regarding how a lane change should behave, including time, travel distance, shape of lane change.
3. Evaluate if the model's lane change can be deemed realistic.