Examensarbete

Automated Fault Tree Generation from Requirement Structures

Examensarbete utfört i Fordonssystem
vid Tekniska högskolan vid Linköpings universitet
av

Johan Andersson

LiTH-ISY-EX--15/4900--SE

Linköping 2015

Automated Fault Tree Generation from Requirement Structures

Examensarbete utfört i Fordonssystem
vid Tekniska högskolan vid Linköpings universitet
av

Johan Andersson

LiTH-ISY-EX--15/4900--SE

Handledare: **Daniel Jung**
ISY, Linköpings universitet
Mattias Nyberg
Scania

Examinator: **Erik Frisk**
ISY, Linköpings universitet

Linköping, 23 oktober 2015

Avdelning, Institution Division, Department Vehicular Systems Department of Electrical Engineering SE-581 83 Linköping	**Datum** Date 2015-10-23

Språk Language ☐ Svenska/Swedish ☒ Engelska/English ☐ __________	**Rapporttyp** Report category ☐ Licentiatavhandling ☒ Examensarbete ☐ C-uppsats ☐ D-uppsats ☐ Övrig rapport ☐ __________	**ISBN** — **ISRN** LiTH-ISY-EX--15/4900--SE **Serietitel och serienummer** Title of series, numbering **ISSN** —

URL för elektronisk version

`http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-XXXXX`

Titel
Title

Automatisk felträdsgenerering från kravstrukturer

Automated Fault Tree Generation from Requirement Structures

Författare
Author

Johan Andersson

Sammanfattning
Abstract

The increasing complexity of today's vehicles gives drivers help with everything from adaptive cruise control to warning lights for low fuel level. But the increasing functionality also increases the risk of failures in the system. To prevent system failures, different safety analytic methods can be used, e.g., fault trees and/or FMEA-tables. These methods are generally performed manually, and due to the growing system size the time spent on safety analysis is growing with increased risk of human errors. If the safety analysis can be automated, lots of time can be saved.

This thesis investigates the possibility to generate fault trees from safety requirements as well as which additional information, if any, that is needed for the generation. Safety requirements are requirements on the systems functionality that has to be fulfilled for the safety of the system to be guaranteed. This means that the safety of the truck, the driver, and the surroundings, depend on the fulfillment of those requirements. The requirements describing the system are structured in a graph using contract theory. Contract theory defines the dependencies between requirements and connects them in a contract structure.

To be able to automatically generate the fault tree for a system, information about the system's failure propagation is needed. For this a Bayesian network is used. The network is built from the contract structure and stores the propagation information in all the nodes of the network. This will result in a failure propagation network, which the fault tree generation will be generated from. The failure propagation network is used to see which combinations of faults in the system can violate the safety goal, i.e., causing one or several hazards. The result of this will be the base of the fault tree.

The automatic generation was tested on two different Scania systems, the fuel level display and the dual circuit steering. Validation was done by comparing the automatically generated trees with manually generated trees for the two systems showing that the proposed method works as intended. The case studies show that the automated fault tree generation works if the failure propagation information exists and can save a lot of time and also minimize the errors made by manually generating the fault trees. The generated fault trees can also be used to validate written requirements to by analyzing the fault trees created from them.

Nyckelord
Keywords

fault tree, Scania, safety analysis, bayesian network, requirement structure

Abstract

The increasing complexity of today's vehicles gives drivers help with everything from adaptive cruise control to warning lights for low fuel level. But the increasing functionality also increases the risk of failures in the system. To prevent system failures, different safety analytic methods can be used, e.g., fault trees and/or FMEA-tables. These methods are generally performed manually, and due to the growing system size the time spent on safety analysis is growing with increased risk of human errors. If the safety analysis can be automated, lots of time can be saved.

This thesis investigates the possibility to generate fault trees from safety requirements as well as which additional information, if any, that is needed for the generation. Safety requirements are requirements on the systems functionality that has to be fulfilled for the safety of the system to be guaranteed. This means that the safety of the truck, the driver, and the surroundings, depend on the fulfillment of those requirements. The requirements describing the system are structured in a graph using contract theory. Contract theory defines the dependencies between requirements and connects them in a contract structure.

To be able to automatically generate the fault tree for a system, information about the system's failure propagation is needed. For this a Bayesian network is used. The network is built from the contract structure and stores the propagation information in all the nodes of the network. This will result in a failure propagation network, which the fault tree generation will be generated from. The failure propagation network is used to see which combinations of faults in the system can violate the safety goal, i.e., causing one or several hazards. The result of this will be the base of the fault tree.

The automatic generation was tested on two different Scania systems, the fuel level display and the dual circuit steering. Validation was done by comparing the automatically generated trees with manually generated trees for the two systems showing that the proposed method works as intended. The case studies show that the automated fault tree generation works if the failure propagation information exists and can save a lot of time and also minimize the errors made by manually generating the fault trees. The generated fault trees can also be used to validate written requirements to by analyzing the fault trees created from them.

Acknowledgments

I would like to start by thanking Scania for giving me the opportunity to do my master thesis at their department RESA in Södertälje, it has really been a great experience. Next I would like to thank my supervisor at Scania, Mattias Nyberg, as well as Anton Einarson, for all their invaluable support and feedback during the master thesis project. I also want to thank the other master thesis student in the project, Oscar Thydén, for always having someone to discuss the thesis with. At LiU I would really like to thank Daniel Jung for the support during the whole thesis, especially with all the help with the report. Last I would like to thank my examiner, Erik Frisk, for all the feedback and help to steer the thesis in the right direction.

Linköping, October 2015
Johan Andersson

Contents

Notation

Abbreviations

Abbreviation	Description
AE	Allocation Element
APPL	Application Layer
ASIL	Automotive Safety Integrity Level
BIOS	Basic Input/Output System
BN	Bayesian Network
CAN	Controller Area Network
CCF	Common Cause Failure
CMS	Chassis Management System
COO	Coordinator
CPT	Conditional Probability Table
DAG	Directed Acyclic Graph
DCS	Dual Circuit Steering
ECU	Electronic Control Unit
E/E	Electronic/Electrical
EMS	Engine Management System
FLD	Fuel Level Display
FM	Failure Mode
FMEA	Failure Modes and Effects Analysis
FSR	Functional Safety Requirements
FTA	Fault Tree Analysis
HWSR/SSR	Hardware and Software Safety Requirements
ICL	Instrument Cluster
NF	No Fault (failure mode)
SESAMM	Scania Electrical System Architecture Made for Modularization and Maintenance
SG	Safety Goal
TSR	Technical Safety Requirements

1 Introduction

1.1 Background and problem formulation

The vehicles of today are very complex and contain a lot of electronics and electrical devices. A modern truck can contain over fifty Electronic Control Units (ECU). Also, every ECU has a lot of electronic components connected to them which results in a large system. The system can fail and components can break. Therefore, a documentation of the failure propagation in such a system is required. Failure propagation means how a fault in a faulty component can effect the functionality of other parts of the systems which can cause failures in other parts of the system, e.g., a sensor sending wrong information can result in faulty results in systems using the sensor data. The fault propagation risks causing a failure on the vehicular level, i.e. failures affecting the whole vehicle, which can cause harm to both people and equipment. However, to manually generate the documentation of such a large system is very complicated and time consuming. Since the systems are constantly evolving, the procedure has to be done over and over again. If fault propagation analysis could be done automatically a lot of time could be saved.

Knowledge about failure propagation in the system is important for several reasons. First of all, it is used in workshops for troubleshooting after a failure has been detected when it is up to a mechanic to locate and fix the problem. More importantly well documented failure propagation is also needed for the safety analysis of a vehicle, where probabilities and effects of possible failures are fundamentally important to prevent hazards, i.e., unwanted events that can cause accidents. Lastly, failure propagation can be used to create FMEA (Failure Modes and Effects Analysis) and fault trees, which are required to fulfil new automotive standards, ISO26262 [2]. A method for describing failure propagation as well as an method for automatically use that information, to create fault trees and FMEA, will assist the safety engineers at Scania.

The research project ESPRESSO at Scania have created underlying architectural models and safety requirements describing the electronic/electrical (E/E) systems and made them available in a database. Safety requirements describe the intended functionality of safety critical components or systems. The functionality is guaranteed to be fulfilled if a group of assumptions on the behaviour of the environment of the component/system are fulfilled.

Each E/E system's main functionality is described by a set of top level safety requirements called safety goals. A safety goal specifies the intended functionality of the system and has to be fulfilled to guarantee that the truck is in a safe state. The safety goal is then further broken down into lower level requirements. The system will be described using a network of connected requirements which is called a requirement structure. An example of a safety requirement from the system dual circuit steering is described in Example 1.1. Dual circuit steering is the system which handles the hydraulic steering system.

Example 1.1

A safety goal (SG) of the system dual circuit steering is as follows:
"If nominal driving, then steering wheel torque applied must make vehicle turn."

The SG has the following assumptions:
SubGoal1 and MechSteeringReq1. SubGoal1 promises that "If nominal driving, then there must be sufficient hydraulic flow" and MechSteeringReq1 guarantees that "If there is sufficient hydraulic flow, then steering wheel torque applied must make vehicle turn".

The requirement (SG) guarantees that if driving in a nominal way the steering wheel will make the truck turn **if** its assumptions are fulfilled. The assumptions are in turn requirements with assumptions of their own. The breakdown of the assumptions will give requirements on smaller parts of the system, which all has to be fulfilled to guarantee the safety goals behaviour. All requirements broken down from the SG is together building a requirement structure.

One task in the thesis is to use the requirements and their structure to find out what additional information that is needed to describe the failure propagation in E/E systems. A method to automatically generate failure propagation networks from that information will be developed. Also, a method to use the failure propagation network to automatically generate fault trees as well as FMEA tables will be developed.

To complete the failure propagation network it is needed to describe how failure modes in one requirement depends on the failure mode in its parents (the no fault mode is also considered a failure mode). In a directed graph the parent is the start point of a directed arc and the child is its destination.

The methods shall be evaluated against two real Scania systems, the fuel level display and the dual circuit steering which, is the systems handling the display showing the fuel volume and the hydraulic steering system.

1.2 Introduction to safety engineering concepts

This section will briefly describe fault trees and FMEA and what purpose they fill in many safety applications. The focus on the thesis is on fault trees, but a short introduction to FMEA will be included as well.

1.2.1 Fault tree analysis

Fault tree analysis is about understanding how unwanted events, called hazards, can occur [20]. Possible hazards are broken down into possible causes, e.g., which failing systems or hardware components that can cause the hazard. The result of an analysis of how faults in the different components will propagate and eventually causing a hazard can be represented as a fault tree. The causes are connected with logical gates that in a compact way will describe what combinations of faults that can cause the hazard. The logical gates are either OR or AND. If it is an OR gate it is enough that one component fails for that part of the tree to fail, while if it is an AND gate all the inputs to the gate must fail for that part of the system to fail.

Fault tree analysis is a top-down process because it starts from a hazard on the top level and is broken down to more basic causes of the hazard [20].

1.2.2 FMEA

Failure Mode and Effect Analysis (FMEA) and is used to find out which hazards a fault in a single component can cause, and which possible effects they can have [19]. The FMEA is a table which will describes which hazardous events, effects on the system, and failure modes the component has. The FMEA also includes information about the probability, severity and detection of the fault, which will give a classification about the quality of the component. The probability states how likely the failure is while the severity states how dangerous the hazards caused by the failure are. The detection tells how likely it is to detect a failure before it causes the hazard.

The FMEA generation is called a bottom-up process since it starts on a low level and goes upwards to the hazards it can cause.

1.2.3 ISO26262

ISO is the International Organization for Standardization and is a federation for standards that stretches worldwide. ISO26262 is, according to [1] an adaptation of Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (IEC 61508) applied to the electrical and/or electronic (E/E) system within road vehicles. This adaptation of IEC 61508 applies to all activities during the safety life cycle of safety-related systems comprised of electrical, electronic and software components.

The need for an international safety standard started to to rise when new functionality in automobile development increasingly begin to enter the domain of safety engineering.

With more complex systems, the risk for system failures or random hardware failures grows. Guidance to avoid these risks is included in the ISO26262 standard by providing standardized processes and requirements.

The goal is to reach a high level of functional safety which is defined as "absence of unreasonable risk due to hazards caused by malfunctioning behaviour of E/E systems". The functional safety is dependent on the the development process such as design, implementation, integration, specification of requirements, etc.

This thesis is based on the upcoming need for the truck industry to comply with this new standard and therefore much of the terminology used in the thesis is taken directly from the ISO26262 standard [1].

For every system an ASIL (Automotive Safety Integrity Level) is defined for all hazards. Depending on how dangerous the hazards are they, are classified from A to D on the ASIL scale, where A dictates the lowest integrity requirements while ASIL D dictates the highest [1]. To fulfill ISO26262 fault trees are needed in systems with hazards of ASIL C or D. Figure 1.1 describes the need for inductive (FMEA etc.) as well as deductive (FTA etc.) safety analysis methods depending on different ASIL levels.

"o" indicates that the method has no recommendation for or against its usage for the identified ASIL while "+" indicates that the method is recommended for the identified ASIL. "++" indicates that the method is highly recommended for the identified ASIL.

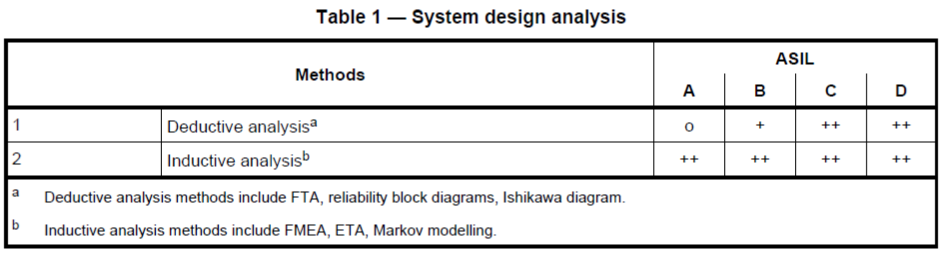

Table 1 — System design analysis

Methods		ASIL			
		A	B	C	D
1	Deductive analysis[a]	o	+	++	++
2	Inductive analysis[b]	++	++	++	++

[a] Deductive analysis methods include FTA, reliability block diagrams, Ishikawa diagram.

[b] Inductive analysis methods include FMEA, ETA, Markov modelling.

Figure 1.1: *Table describing when deductive and inductive methods are needed to fulfill ISO26262[2]*

.

1.3 Purpose and goal

The primary goal of this master thesis project is to develop a method that automatically generates fault trees (FT) based on the available requirement structure stored in a database. The requirement structure will be transformed into a Bayesian network describing the requirement structure as well as an additional layer that describes what failure mode the requirements will be in depending on which fault/faults that occurred in the system. The fault trees might need more information than the requirement structure, and one goal is to investigate what that information is, e.g., failure mode propagation.

The fault tree generation tool shall also be evaluated. The evaluation is conducted partly by interviewing Scania employees responsible for the systems that are modeled by the generated fault trees. Evaluation is also done by manually creating the fault trees from the requirements and comparing the results to the generated trees. The accuracy as well as the usefulness of fault trees at Scania will also be investigated, e.g., to fulfil the automotive standard ISO26262. Finally the fault tree generation shall be integrated into an existing tool chain at Scania.
The goals of the thesis in conclusion are:

1. Develop a method to automatically generate fault trees based on data describing systems, called requirement structures.

2. Investigate what information is needed to generate a failure propagation network and use it to generate a fault tree.

3. Investigate how to automatically generate the failure propagation network.

4. Evaluate the generated fault trees on two systems together with Scania experts.

This thesis focuses on generating fault trees and is done in collaboration with another master thesis project about generating FMEA from the same information. Parts that are used in both theses are the generation of Bayesian networks that will be used to modulate the failure propagation in systems, as well as the method made for designing the failure propagation in every requirement node.

1.4 Limitations

A big limitation for the thesis is that there didn't exist requirements for more then two Scania electronic/electrical systems, which limit the possibility to test and verify the fault tree generation method.

Another limitation is that Scania currently doesn't use any deductive methods mentioned in [3], i.e. fault trees in the safety analysis. This limits the possibilities to verify the results since there exists no other fault trees to compare with. Instead the generated fault trees will be validated with help from people at Scania who are experts of the analyzed subsystems.

Also, in this thesis project , the investigation consider static fault trees, i.e., they do not include dynamic gates which are used to model more complex system behaviour [6]. As well as all events in the fault trees are assumed to be statistically independent.

1.5 Related research

The first intended part of the master thesis project will focus on how to use safety requirements to create Bayesian Networks (BN) containing failure propagation. The article [17] mentions how to generate a Causal BN containing failure propagation from safety requirements.

The article [25] gives an understanding how the requirements are defined and structured using contract theory. The requirement structure described here is the foundation that this thesis is based on.

Paper [24] explains how to apply the ISO26262 standard to break down a safety goal into software and hardware requirements as well as introducing the concept of contracts.

One approach how to create Bayesian networks from fault trees is mentioned in [4]. The inverse method is used in this thesis and it is interesting to compare the usage to see how the different parts of the fault tree is transformed into nodes in the Bayesian network. This transformation can be partly reversed when creating fault trees from Bayesian networks.

Faults in a system and how the failure mode can change when propagating through a system is described in [21]. However, the system structure in this article is built around signals and components instead of requirements. Differences in structure makes the methodology hard to translate, but it is still interesting to see examples of how different faults can translate or propagate. [5] is another article about how to transform a fault tree into a Bayesian network.

Paper [15] describes a way to modulate failure propagation in a complex system. This article focus on how the increased load from a failing part of the system propagates and risk causing more failures, instead this thesis focus more on how failure propagation throughout the system can affect the safety.

A way of automatically generating fault trees is presented in [14]. The difference between this thesis and the article is that in the article the model of the fault tree is taken from a technical process description in this thesis the fault trees are modelled from requirements and the structure they form. The system model in [14] is injected with failures and the fault tree is generated from that data.

The part of this master thesis project that is new compared to other found research, is in the way that the automatic creation of fault trees is based on requirement architecture and requirements. With respect to previous works, the goal here is to automatically generate fault trees from safety requirements.

1.6 Methodology

This master thesis is made as a case study to investigate if it is possible to generate fault trees solely from a requirement structure and if not investigate what additional information that is needed to do so.

The first part of the thesis concerning the implementation, is based on the articles [17] concerning using a Bayesian network to create a failure propagation network. The data of the fuel level display system to be used, is based on a requirement structure described in [25]. The requirement data used for dual circuit steering is taken from a example used in a demonstration [23]. The mentioned data is computerized and put into a database, further described in Section 3.4. The data is then used to create a failure propagation network together with some assumed information about how failure modes propagate between requirements, which then was used to create fault trees. Validation of the generated fault trees is done by manually generating trees and comparing them to the automatically generated trees. The fault tree for the system fuel level display is also validated by creating a fault tree of the system together with Scania experts.

1.7 Report outline

The outline of this master thesis report is as follows:

In Chapter 2, the basic theory and information behind the project is presented. In Chapter 3 a system overview for the two systems, fuel level display and dual circuit steering, are described as well as a short information about the database is presented.

Chapter 4 describes the method that is used to generate failure propagation networks presented in the form of a Bayesian network. Chapter 4 also describes the method for automatic fault tree generation.

Evaluation of the generated trees are done in Chapter 5 by comparing the generated fault trees to manually created trees. In Chapter 6 the results of the thesis are presented, i.e. the created propagation graph and the fault trees are discusses and their quality is verified. In Chapter 7 a discussion about the thesis and conclusions are from it are drawn and described. A couple of future works are also suggested to continue this work.

2 Theory

The theory chapter starts with a short description on the ISO26262 standard which motivates this thesis. The chapter mentions some general terminology which will be used throughout the report.

The theory behind the requirements is presented in Section 2.2 followed by theory about Bayesian networks as well as Boolean algebra. Section 2.4.2 describes how fault trees are created and analysed.

2.1 General terminology

In this section, general terminology that will be used throughout the report is defined. The definitions are taken directly from [1].

Definition 2.1 (Element). System or part of a system including components, hardware, software, hardware parts and software units.
[1, p. 6]

Definition 2.2 (System). Set of elements that relates at least a sensor, a controller and an actuator with one another.
[1, p. 17]

Definition 2.3 (Item). System or array of systems to implement a function at the vehicle level, to which ISO 26262 is applied.
[1, p. 10]

Definition 2.4 (Fault). Abnormal condition that can cause an element or an item to fail.
[1, p. 7]

Definition 2.5 (Failure). Termination of the ability of an element to perform a function as required.

[1, p. 7]

Definition 2.6 (Failure mode). Manner in which an element or an item fails.
[1, p. 7]

Definition 2.7 ((Functional) Safety requirement). Specification of implementation-independent safety behaviour, or implementation-independent safety measure, including its safety-related attributes.
[1, p. 8]

Definition 2.8 (Hazard). Potential source of harm (physical injury or damage to the health of persons) caused by malfunctioning behaviour of the item.
[1, p. 9]

Definition 2.9 (Safety goal). Top-level safety requirement as a result of the hazard analysis and risk assessment.
NOTE: One safety goal can be related to several hazards, and several safety goals can be related to a single hazard.
[1, p. 14]

Definition 2.10 (ASIL). One of four levels to specify the item's or element's necessary requirements of ISO 26262 and safety measures to apply for avoiding an unreasonable residual risk with D representing the most stringent and A the least stringent level.
[1, p. 2]

2.2 Contract theory

Contract theory is used to create the requirement structure that is the data that the failure mode propagation network is based on. The theory in this chapter is mainly based on the works presented in [22] and [25].

2.2.1 Background

The concept of contracts were first introduced in formal specification of software interfaces were it was specified as pre- and post-conditions. The concept later developed to work as a design philosophy when designing Cyber-Physical Systems. Cyber-Physical Systems embody the interactions between physical objects and computers according to [10]. Typically Cyber-Physical Systems consists of a system of devices that performs physical actions as well as the computers that controls them.

Contract theory is used in ESPRESSO, a Scania project, in a case study to structure and specify safety requirements in ISO26262, which is described in the article [26]. A contract is defined as a guarantee as well as a set of assumptions. The guarantee promise a specified result or behaviour if the assumptions are fulfilled. This can be used when structuring ISO26262 safety requirements, since a safety requirement is a intended behaviour of an element in a system. If the safety requirement is set as a guarantee of a contract some conditions can be set which has to be fulfilled for the safety requirement to hold. In ISO26262 a safety requirement has to be allocated to an element in the system, and therefore the contract will be allocated to an element in structure, meaning that nominal behaviour of the element is a requirement for the contract to be fulfilled. Also a set of assumptions can be connected to the contract where the assumptions can be that other safety requirements in the surroundings are fulfilled, i.e., other guarantees has to be fulfilled. Safety requirements can be connected to each other in this way. This is shown in Figure 2.1. In the figure G is the guarantee of contract 2 and A1, A2 as well as G are Gs assumptions. G is also the guarantee of contract 1, which has the assumptions A1, A2 and A3.

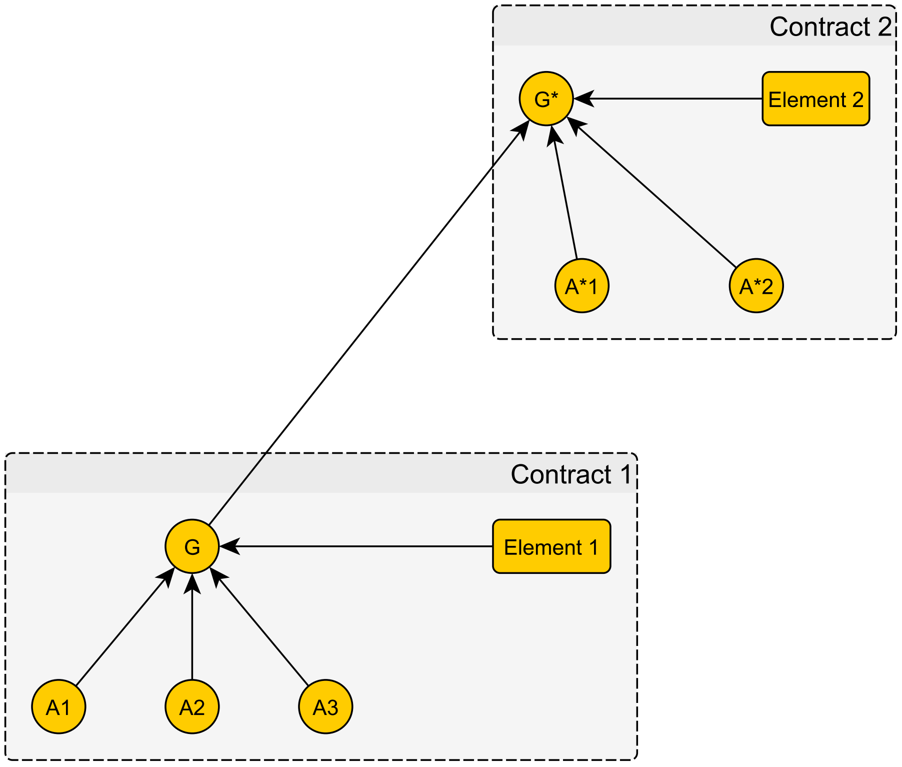

Figure 2.1: *An example of two contracts, in which G stands for guarantee and A for assumption. Here, the guarantee of the first contract is an assumption in the second contract.*

If an assumption is not fulfilled neither can the guarantee be fulfilled. If the guarantee is an assumption of another contract, it will result in another broken contract. This behaviour means that connected contracts are depending on each other. If starting from a safety goal, see Definition 2.9, a system can be broken down to lower level requirements and using contract theory the whole system can be connected in a so called requirement (contract) structure. A requirement structure of the system fuel level display is shown in Figure 2.2.

2.2.2 Definitions

In this section the contract theory definitions that are used, when creating the structuring the safety requirement, is presented. The definitions are taken from the articles [25] and [22].

Definition 2.11 (Run). Let $X = \{x_1, ..., x_N\}$ be a set of variables. Consider a pair (x_i, ξ_i) of a variable x_i and a trajectory ξ_i of values of x_i over a time window of possibly infinite length, starting at a certain time t_0. A set of such pairs, one for each variable in X is called a run for X, denoted ω_X.
[25, p. 3].

A run for a set of variables is a trajectory in each of the variables. The trajectories of the variables can be limited by using assertions, see Definition 2.12, on the variable set.

Definition 2.12 (Assertion). Given a set of variables X' and a time window, an assertion W over X' is a possibly empty set of runs for X'. An assertion W can be specified by a set of constraints, e.g., equations, inequalities etc.
[25, p. 4]

A set of constraints, e.g., equations, inequalities etc. can be used to specify an assertion. This means that an assertion can, for example, be the set of all possible runs that fulfill a specified inequality or equation.

Definition 2.13 (Element). An *element* $\mathbb{E}$ is an ordered pair (X,B) where:
a) X is a non-empty set of variables, called the *interface* of $\mathbb{E}$ and where each $x \in X$ is called a *port variable*; and
b) B is an assertion over X, called the *behaviour* of $\mathbb{E}$.
[25, p. 5]

An element is a model of things such as software, hardware, or physical entities. Can be compared to Definition 2.1 in ISO26262.

Definition 2.14 (Architecture). An *architecture* $\mathscr{A}$ is a set of elements organized into a rooted tree. [25, p. 7]

The architecture is a way to structure elements in order to model a Cyber-Physical system as well as it's surroundings.

Definition 2.15 (Contract). A *contract* is a pair (A, G), where i) G is an assertion, called guarantee; and
ii) A is a set of assertions $\{A_i\}_{i=1}^{N}$ where each A_i is called an *assumption*.

In the context of an architecture, a guarantee of a contract for an element expresses an intended property under the responsibility of the element, given that the environment of the element fulfills the assumptions. [25, p. 8]

Definition 2.16 (Requirement). A requirement is an assumption or a guarantee
[25, p. 16]

The guarantee of a contract, allocated to an element $\mathbb{E}$ assures that the intended property of the element is fulfilled, given that the assumptions on the element's environment as well as the behaviour of the element are fulfilled.

The guarantee gives the allocated element the responsibility for an intended behaviour to be fulfilled, given that the assumptions, i.e., the environment, are fulfilled. [25]

Definition 2.17 (Requirement structure for architecture). Given an architecture $\mathscr{A}$ and a set $(\mathscr{A}_{k,l}, G_{k,l})_{j=1}^{N}$ where $(\mathscr{A}_{k,l}, G_{k,l})$ is a contract for an element $\mathbb{E}_i$ of $\mathscr{A}$ where each assumption in each set $\mathscr{A}$ is either

a) a guarantee of a contract for a sibling of $\mathbb{E}_i$;

b) an assumption of a contract for a proper ancestor of $\mathbb{E}_i$;

c) an assumption of the contract for the root element in $\mathscr{A}$,

then a requirement structure $\mathscr{C}$ for $\mathscr{A}$ is an arc-labeled Directed Acyclic Graph (DAG), such that:

i) the guarantees $G_{i,j}$ and the assumptions of the contract for the root element in $\mathscr{A}$ are the nodes in $\mathscr{C}$;

ii) each arc is uniquely labeled either ”Assumption of” or ”Fulfills”;

iii) there is an arc labeled ”Assumption of” from a node W to $G_{i,j}$, if and only if W is in $A_{i,j}$;

iv) if there is an arc labeled ”Fulfills” from $G_{i,j}$ to $G_{k,l}$, then $G_{k,l}$ is a guarantee of a contract for a proper ancestor of $\mathbb{E}_i$ and

v) if a guarantee $G_{k,l}$ is reachable from an assumption A of a contract for a proper ancestor $\mathbb{E}_m$ of $\mathbb{E}_i$, then A is also an assumption of any contract ($\mathscr{A}_{k,l}$, $G_{k,l}$) where $\mathbb{E}_k$ is a proper ancestor of $\mathbb{E}_i$ and a descendant of $\mathbb{E}_m$ (including itself) and where $G_{k,l}$ is reachable from $G_{i,j}$.

v) if a guarantee $G_{i,j}$ is reachable from an assumption A of a contract for a proper ancestor of $\mathbb{E}_m$, then A is also an assumption of any contract $(A_{k,l}, G_{k,l})$ where $\mathbb{E}_k$ is a proper ancestor of $\mathbb{E}_i$ and a descendant of $\mathbb{E}_m$ (including itself) and where $G_{k,l}$ is reachable from $G_{i,j}$.

[25, p. 15]

In Figure 2.2 a requirement structure for the system fuel level display is shown. In the figure the nodes are contracts and incoming arcs denoted with an arrow or a black dot, where arrows are the "Fulfills" relation and and black dots are assumptions to the contract. In the subscript of the name the allocated element is mentioned, i.e., the contract FSR_{Driver} is allocated to the element Driver in the architecture. The requirements in the structure are the guarantees of all contract nodes.

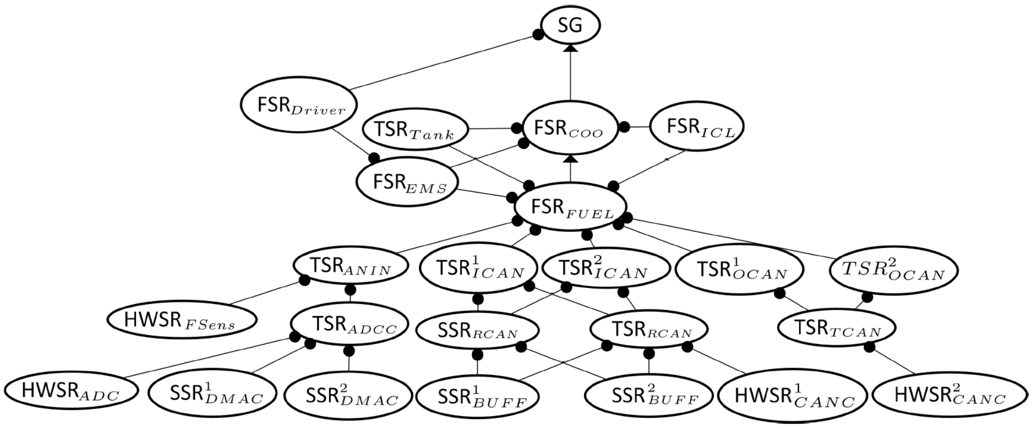

Figure 2.2: *Requirement structure over the fuel level display. From the article [25].*

Definition 2.2.2 gives restriction of how requirements can be structured, e.g. the nature of assumptions and what the arc between requirements means. There are two relationships between requirements. The one that says *assumption* means that the guarantee of a contract only can be fulfilled if all the assumptions are fulfilled. The other relationship is called *Fulfills*. If a requirement R* has the arc *Fulfills* to a requirement R it means that the requirement R* is a proper subset of the requirement R. Example 2.18 describes the relationship *Fulfills* further.

Example 2.18

If one requirement R has a lower level requirements, R*, which fulfills R. It means that R* overtake the responsibility that a subset of R's guarantee is fulfilled.

It can be seen as R* is a subsystem of R, e.g., if R is an ECU responsible for a behaviour B then R* can be a software part of the ECU that is responsible for that specific behaviour.

Definition 2.19 (Goal in Requirement Structure). A node in a requirement structure for an architecture is a *goal* if the node does not have any successors.
[25, p. 16]

A goal from Definition 2.19 is equivalent to the ISO26262 definition of safety goals, see Section 2.1. When structuring safety requirements with ISO26262 [25] the safety goal is set as the guarantee of the goal in the requirement structure.

Definition 2.20 (Extension of Requirement Structure). An extension of the requirement structure defined in Definition 2.2.2.

i) Each node is either a requirement $R_{i,j}$ or an '$OR_\perp$' node and each requirement $R_{i,j}$ is a node;

ii) If and only if $R_{i,j}$ is an assumption of $R_{k\neq i,l}$ then there exists an arc labeled "assumption of" from $R_{i,j}$ to either: a) $R_{k\neq i,l}$; or b) an '$OR_\perp$' node that has exactly one outgoing arc to $R_{k\neq i,l}$, labeled "assumption of";

iii) If an '$OR_\perp$' node has an incoming arc labeled "fulfills" from $R_{i,j}$, then the '$OR_\perp$' node has exactly one outgoing arc to a requirement $R_{k \neq i,l}$ on the parent of $\mathbb{E}_i$;

iii) Each '$OR_\perp$' node has at least two incoming arcs and where any two incoming arcs to the '$OR_\perp$' node, are requirements on different elements

[22, p. 6]

2.3 Bayesian networks

Information about Bayesian networks described in this section has been taken from [11].

Bayesian networks use the concept of conditional probability, and Bayes' rule.

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \tag{2.1}$$

By using Bayes' rule it is possible to update the beliefs of an event X given new information about the event Y has been observed. The updated probability of X, $P(X|Y)$ is called the posterior probability of X while $P(X)$ is called the prior probability of X [11].

Bayes' rule can be extended to

$$P(X|Y,Z) = \frac{P(Y|X,Z)P(X|Z)}{P(Y|Z)} \tag{2.2}$$

It is also important to note that two events can be independent of each other, e.g., as used in this thesis a fault and a hazard can be independent from each other if the fault can not cause the hazard (or the hazard can not be caused by the fault, depending on direction). Independence is written in the following way:

$$P(X|Y) = P(X) \tag{2.3}$$

The Bayesian relations can also be described in a graphical way, where nodes has a set of states as well as a probability of being in the different states. A node is connected to other nodes with directed arcs that represent conditional probability. The network is built up as a directed acyclic graph, which means that all arcs have a direction and there exist no cycles in the graph. The direction of the arc says which node that is depending on which. If a node X has an incoming arc from node Y, X is called a child of Y, and Y a parent of X. It is possible for a node to have many parents as well as many children.

2.3.1 Conditional probability tables

The Bayesian networks in this thesis use so called conditional probability tables (CPT:s) to describe the probability of an event given information of it's connected nodes. An example of a CPT is shown in Table 2.1. In the table, each row resemble a state of the current node, in the example CPT every column is a combination of failure modes of the parents of the node called Child. Every column defines the probability to be in that state given the combination of states in the parent nodes, e.g., according to Table 2.1, if Parent2 has the failure mode FM1 and Parent1 has FM2 it is 50% probability of Child of having the failure mode FM1 and a 50% chance of having FM2.

In this master thesis, the Bayesian network is used to describe what failure mode exists in a requirement (a node) depending on the failure modes in the requirements assuming the requirement (the nodes parent nodes), as well as the failure mode in the element allocated to the requirement. Note that the failure mode in nodes can be the no fault failure mode (NF) which resembles a fault free state.

An example of a requirement node(child node) with two parent nodes connected to it is shown in Figure 2.3.

Example 2.21

The requirement node and the two parent nodes all have 3 failure modes, the no fault failure mode (NF), failure mode 1 (FM1) and failure mode 2 (FM2). The conditional probability table of the requirement node is shown in Table tabell 2.1.

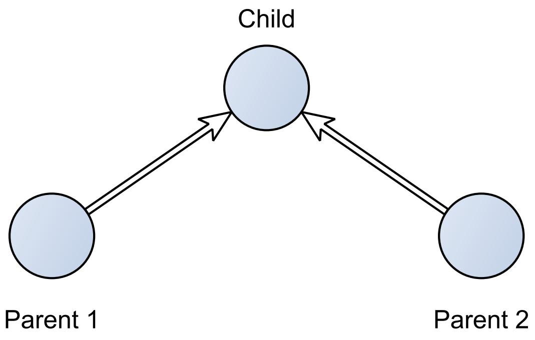

Figure 2.3: *An example of a Bayesian network with three nodes.*

Table 2.1: *Example of a conditional probability table that describes conditional probabilities at a node in the Bayesian Network*

Parent2	NF			FM1			FM2		
Child \ Parent1	NF	FM1	FM2	NF	FM1	FM2	NF	FM1	FM2
NF	1	0	0	0	0	0	0	0	0
FM1	0	1	0	1	1	0.5	0	0.5	0
FM2	0	0	1	0	0	0.5	1	0.5	1

2.3.2 Conditional probability tables size

A big downside with conditional probability tables is that they grow fast with the number of states of the node as well as the number of parents and parent states.

Take an arbitrary conditional probability table describing failure mode propagation, A, describing a node N. A has n rows and m columns, where n is the number of failure modes of the node

$$n = \#FM(N) \tag{2.4}$$

and m are the number of failure mode combinations of the parents

$$m = \prod_{i \in N_{parents}} \#FM(i) \tag{2.5}$$

This means that the number of elements in the matrix depends on the amount of failure modes of the given node and the amount of combination of the states of the child nodes.

$$\#Elements_{CPT} = n * m \tag{2.6}$$

If all nodes has the same amount of failure modes the equation can be written in the following way

$$\#Elements_{CPT} = n^{b+1} \tag{2.7}$$

where n is the number failure modes and b is the number of parents connected to the node. The matrix A is stochastic which means that the sum of all values in a column are equal to one. Which can be seen in (2.8)

$$\sum_{i=1}^{n} A_{ij} = 1 \quad \forall j \tag{2.8}$$

Each A_{ij} resembles the probability of propagation to failure mode i given the failure modes combinations of the parents, represented in the column j.

2.4 Fault tree analysis

The concept of fault trees and what they consist of as well as the the method of creating them is presented in this section. The section also describes how basic analysis of trees are conducted as well as some limitation with fault tree analysis.

2.4.1 GeNIe and Smile

To create Bayesian networks, SMILE (Structural Modeling, Inference, and Learning Engine) is used [13]. SMILE is a library of C++ classes containing implementations of decision-theoretic methods, e.g., Bayesian networks. GeNIe [13], which is SMILEs graphical interface, can be used to visualize Bayesian networks created using SMILE. SMILE is used to define conditional probability tables as well as calculate the inference between nodes, while GeNIe can be used to visualize the failure propagation networks.

2.4.2 Fault trees

Fault tree analysis (FTA) is a deductive method used in safety, risk and reliability analysis. FTA is used to try to describe how an undesired event can happen, and what the possible combination of faults that can cause it.

A fault tree consist of following parts:

1. Top event. The undesired event that is analysed with FTA can be, e.g., a system failure.
2. Basic events, basic causes of the top event, e.g., a broken sensor or actuator. Basic events are atomic parts of the fault tree.
3. AND/OR-gates. Logical gates that connects basic events or branches of the tree together.
4. Intermediate events. Intermediate events are located over each gate in the tree and is the event describes by the gate beneath it. Can be, e.g., a cluster of components or a subsystem.

An example of a fault tree, trying to explain why a laptop won't start, can be seen in Figure 2.4. The example system says that for the laptop not being able to start it is sufficient that the operating system doesn't work correctly, or that any of the listed hardware components is broken. When it comes to not enough power to the laptop, both the battery has to be discharged as well as the computer not being connected to the power outlet.

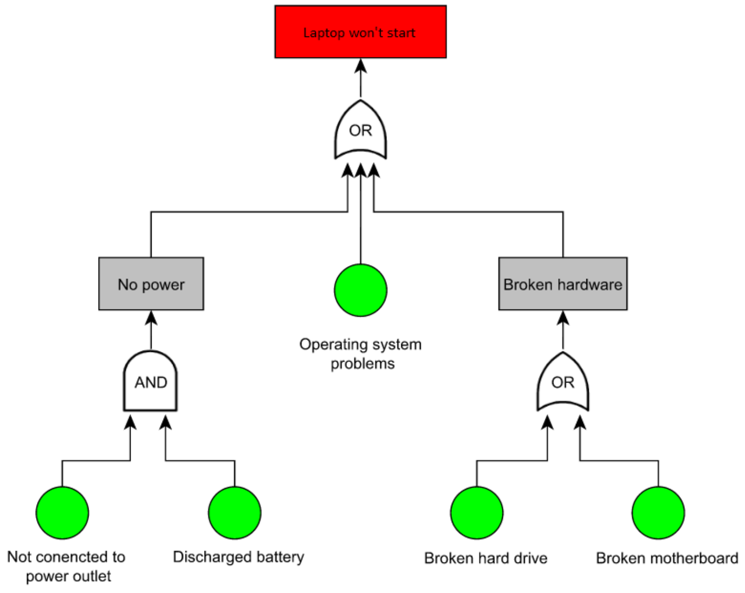

Figure 2.4: *This figure shows an example of a fault tree describing reasons why a laptop wont start.*

2.4.3 Creating fault trees

FTA is a deductive method, which means that it starts from the top and goes down. The first part is to start with an undesired event, often a failure on vehicular level, from the failure deduces the possible causes for that event. The failure is then put in top of the fault tree, and is called the top event [20].

The immediate causes of the top event are then identified. The immediate events are then decomposed into causes of higher resolution by combining basic event or branches with a logic gate. Higher resolution means that the elements are further broken down to a lower level, e.g., instead of having a computer as a basic cause, the computer can be decomposed into the parts the computer consists of. The process often continues until a predetermined resolution of causes has been reached. The resolution of causes can be everything from subsystem level to reaching functions within software components. The next step is to find all basic events causing the top event and existing within the resolution of the tree. When all basic events responsible of causing the top event has been found the basic events are connected - together with logical nodes, e.g., AND and OR-gates - in a graph.

A few assumptions, in accordance to the article [4], have been made when creating the fault trees:

1. Basic events are binary e.g. faulty/not faulty.

2. All events are assumed to be statistically independent.

3. Events are connected using OR and AND gates.

Assumptions 1 and 3 are based on the fact that the contracts are binary, e.g., fulfilled/not fulfilled and that contracts are depending on other contracts to be fulfilled. Which can be described using AND/OR gates. The assumptions that all events are independent is made according to the limitations of the thesis.

The different gates, OR and AND are used to define what different combinations of faults will propagate further up in the net or not. If it is an OR-gate it is enough that one of the incoming arcs, i.e., possible failure causes, is faulty for that part of the system to be faulty. For an AND-gate all of the connected failures have to be faulty for the system to break down. When it is an AND-gate it is said that that part has redundancy which means that, e.g., one of the basic events/subsystems are sufficient for the given part of the system to work as intended.

2.4.4 Analysis

The main reason for fault tree analysis is to find out which combinations of errors or faults in the system that can cause a hazard. The fault tree is analysed to find the minimal set of causes for the hazard, i.e. the minimal ways the top event in the tree can be reached.

Fault trees are mainly used to determine the reliability of a system, by for example pointing out all single-point failures (a failure that single-handed can cause the top event). FTA can also be applied to both existing systems and systems under design. Identifying weaknesses in the system design can be used to help redesigning a system and preventing hazardous events.

Instead of a fault tree, a so called success tree can be created [20]. The success tree is an optimistic version of a fault tree and describes which components has to work for the system to work as intended. Because success and failure are complementary the transformation between the different trees is simple. A first step is to complement the top event, i.e., describing reasons for the undesired event NOT to occur. The basic events are also replaced with their complementary events, i.e., fault free components. Last step is turning all OR-gates into AND-gates and all AND-gates into OR-gates. The complement to a minimal cut set (in a fault tree) is a minimal path set, which describes minimal sets of which basic events that has to work for the system to be failure free, i.e., all the possible ways to assure the top event to not occur [20].

It is often more practical to see a system from a failure perspective in safety analysis. The main reason for using the failure perspective is that it is easier to define a failure state then a success state, e.g., the engine breaks down compared to defining the engine works successfully which can be viewed a little bit more subjectively.

Fault trees are qualitative models of systems and gives information about causes of undesired events, but fault trees can also be used as a quantitative tool if probabilities are introduced. If the probabilities of faults in the basic events are known then the tree can be used to calculate the probability of the top event [20]

3 System overview

The first part of this chapter describes the Scania systems used in this thesis. This is followed by sections describing the existing requirement structures and is finished with how the requirement data is stored in a database as well as an introduction to the software used to generate Bayesian networks.

3.1 Fuel level display

The fuel level display is described in this section. First the system is described then difficulties, when it comes to generate fault trees, with the system are mentioned.

3.1.1 Fuel level display system description

The system information is taken from the chapter in Computer Safety, Reliability, and Security in [25]. The fuel level display (FLD) is the system that estimate the fuel volume and shows it to the driver on a display. It also shows a warning if the fuel volume drops below a certain value. The FLD consists of a few ECU-systems. An ECU-system contains the ECU as well as sensors and actuator connected to it.

The ECU-systems that FLD consists of are the Chassis Management System (CMS), the Engine Management System (EMS) and the Instrument Cluster (ICL). A description of the item of fuel level display, the CMS ECU-system, can be seen in Figure 3.1. The figure is found in the article [25].

The CMS, estimates the current fuel volume in the tank using a Kalman filter. For the Kalman filter to work properly it relies on the measurement signal from the tank as well as an estimation of the current fuel consumption rate from the EMS. The measurement signal is calculated from a floater in the fuel tanks position. A CAN signal containing the estimated fuel volume is sent to the ICL where it together with a low fuel warning, if the fuel level is below 10 %, is displayed to the driver.

The system has two hazards. One of the hazards is **NoFuelLevelWarning** which occurs when no there is no warning showing that the fuel level is low when driving the truck. The other one is the more serious hazard **OutOfFuel** which occurs when the truck runs out of fuel when driving [12].

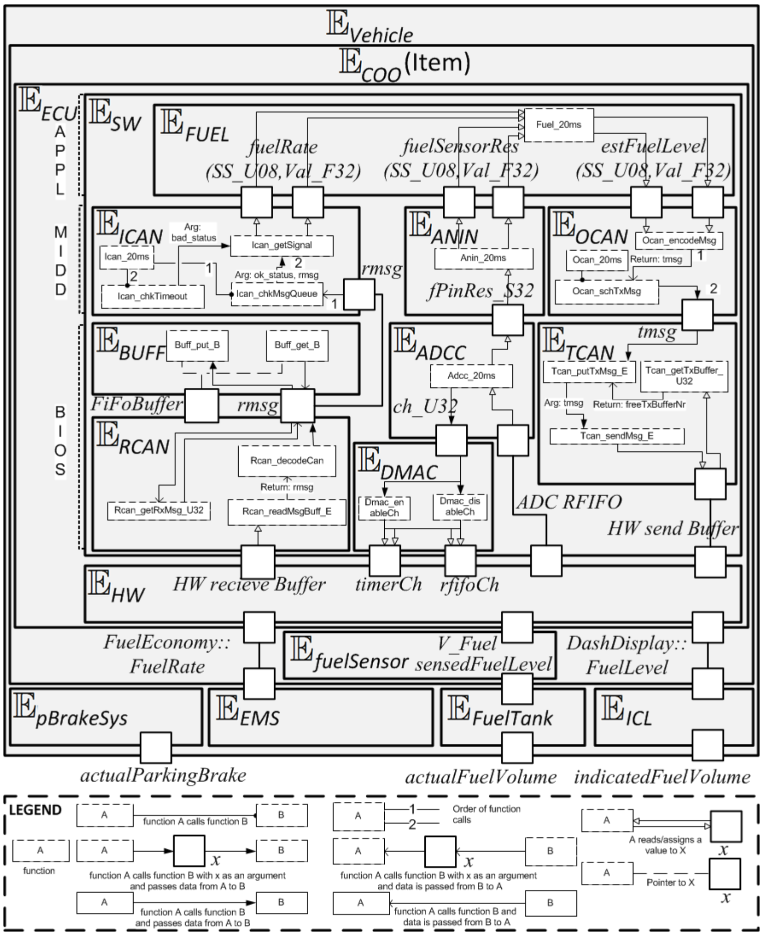

Figure 3.1: *A figure showing the elements that fuel level display contains and its environment. From article [25]*

3.1.2 Fuel level display system difficulties

A problem with fuel level display when it comes to fault tree generation is that the system lacks some kind of back-up system, i.e., there are no redundant subsystems. This means that the whole system will be described as a tree with OR-gates. The system has redundancy in the way that is has error handling. According to [8], static fault trees are bad at describing systems with error handling. Since it was not a trivial problem it was considered to be outside the scope of this thesis. Another system was used as well to validate the fault tree generation process. The other system is the dual circuit steering which is explained in Section 3.2.

3.2 Dual circuit steering

The system dual circuit steering is presented and described in this section. Then, some of the difficulties with the system are highlighted.

The dual circuit steering (DCS) is divided in the same way as Section 3.1 about the fuel level display; where the system is presented in the first part and difficulties is described in the the second part.

3.2.1 Dual circuit steering system description

Dual circuit steering is the system responsible for the power steering of the truck, which helps the driver steer the truck without using considerable effort. The DCS system is split into two independent circuits connected to the steering systems. This gives the system redundancy, which the fuel level display lacked. The safety goal of the dual circuit steering system is: "If nominal driving, the steering wheel torque applied must make vehicle turn". It is very important that the safety goal is fulfilled to guarantee the safety of the driver and other road users. The safety goal is then broken down into sub goals responsible of fulfilling the safety goal. A figure describing the requirement breakdown is shown in Figure 3.3.

In the requirement structure the safety goal is split into two sub-requirements, one requirement concerning the mechanics behind the steering and the other one sub goal requiring sufficient hydraulic flow to the power steering which will help the truck turn.

The hydraulic system is divided into two sub systems: one containing the primary hydraulic circuit, which has the main responsibility to generate enough hydraulic flow, and the other is the back-up circuit, which gives sufficient hydraulic flow for a short period of time if the primary circuit fails. Also, if the primary system fails a warning light notifying the driver there is something wrong with the hydraulic flow will be shown, telling the driver to stop.

The hazard of dual circuit steering is occurring when turning the steering wheel does not make the truck turn, and is called **NoServoSteering** [12].

3.2.2 Dual circuit steering system difficulties

The difficult part of dual circuit steering is the fact that there exists some hardware redundancy in the system. The system has two hydraulic flow circuits, where the back up is used when the primary circuits fails.

The problem with redundancy is that the requirements usually fails if only one of the assumptions fails. In general a contract is only fulfilled if all the assumptions are fulfilled, therefore some extension of contract theory is described in [22]. Definition 2.20 explains what the extension looks like and is based on adding an addition node type in the requirement structure, called $OR_{\perp}$. The new node works as a AND node when it comes to failures which means that all incoming arcs has to fail for the node to fail.

3.3 Requirement structure

Requirement structures used here is based on the theory from [25] which is presented in Section 2.2. When creating a requirement structure, the first step is to identify a top level safety requirement for the system, i.e., a safety goal (SG). When defining the guarantee of a contract, the intended responsibility of the element, in a larger context, is considered.

The safety goal of the fuel level display states that it should not give the driver wrong information about the fuel level, i.e., it should not show the driver that it is more fuel in the tank than there actually is. If the safety goal is violated with the failure mode *signal_value_high* the hazard OutOfFuel occurs.

In the fuel level display there are other levels of requirements than safety goals, which is the highest. The other levels are Functional Safety Requirements (FSRs), Technical Safety Requirements (TSRs), and Hardware and Software Safety Requirements (HWSRs/SSRs) and are referred to depending on what properties they modulate.

If a requirement modulate vehicular level properties, it is considered to a FSR. If signals with both SW and HW properties are referenced by the requirement, it is considered to be a TSR. If only SW/HW properties is modeled it is a HWSRs/SSRs [25].

Figure 3.2 shows the requirement structure of fuel level display the safety goal. The arc with a black dot, in the figure, from FSR_{Driver} to SG declares that SG assumes FSR_{Driver}. This is equivalent to FSR_{Driver} being the assumption to the contract ($\{FSR_{Driver}\}$, SG) where SG is the guarantee. In Figure 3.2, an arrow from FSR_{COO} to SG means that FSR_{COO} fulfills SG.

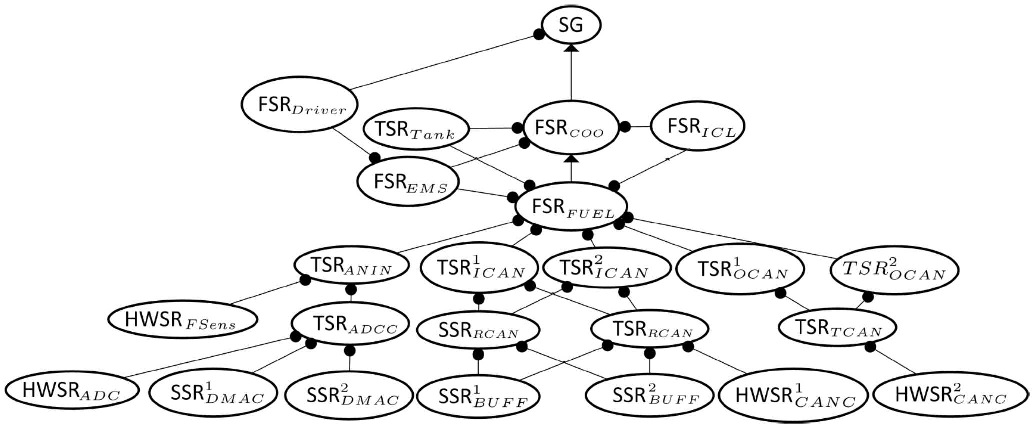

Figure 3.2: *Requirement structure over the fuel level display. The black ball in the end of the arcs means assumption and the black arrow means that a requirement is fulfilled by another requirement. From article [25]*

All of the safety requirements of the dual circuit steering are presented in Figure 3.3. In dual circuit steering, the requirements have not been divided into levels like FSR, TSR, SSRs, and HWSRs. Instead, they are divided into levels from SESAMM function to Allocation Elements (AE) down to infrastructure requirements.

The safety goal of DCS is, as opposed to FLD, divided into several sub goals which is characterized by not being allocated to any element in the architecture. The hardware redundancy described in Section 3.2 about the system is described with a label named 'OR' in Figure 3.3.

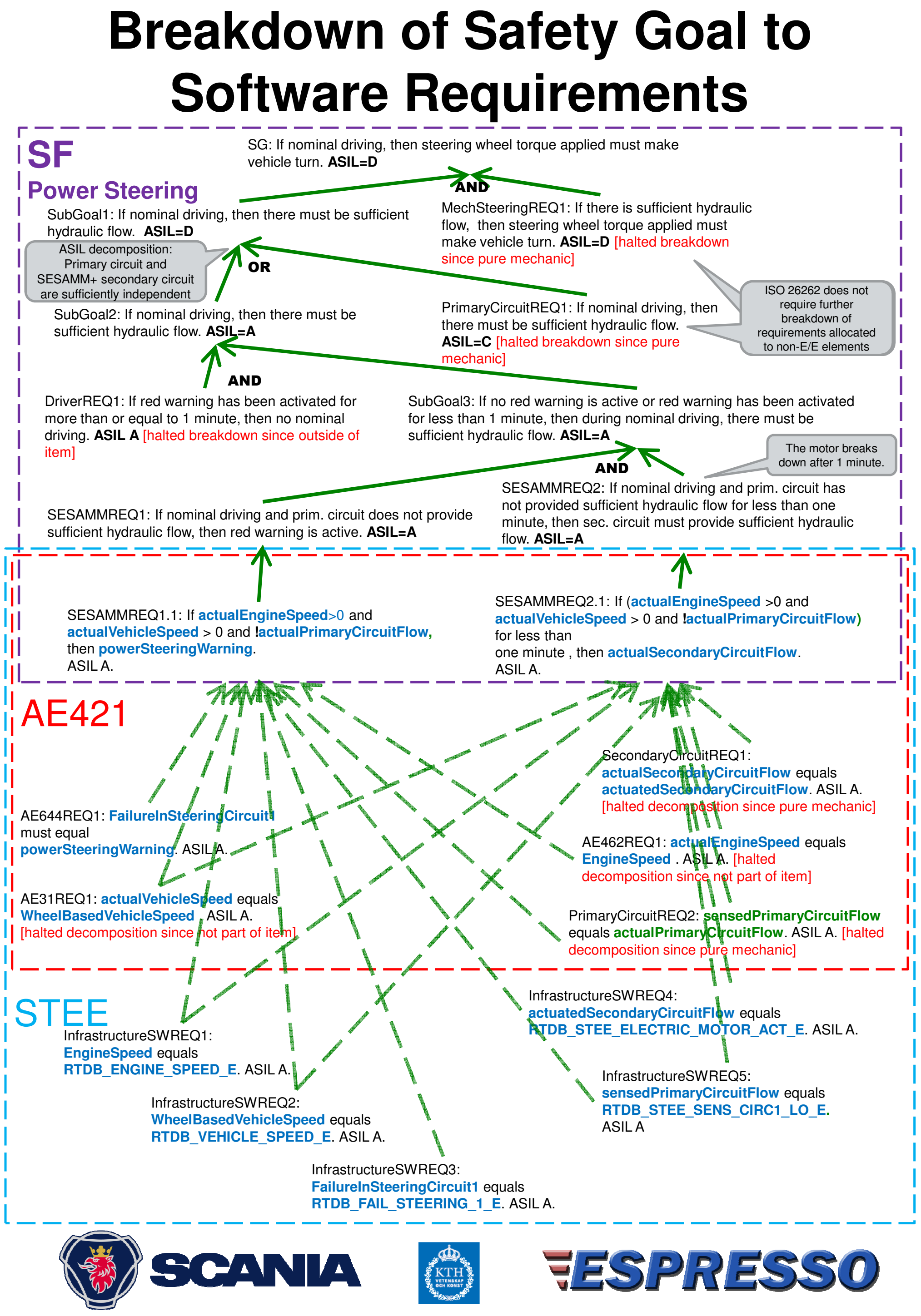
Breakdown of Safety Goal to Software Requirements
SF
Power Steering
SG: If nominal driving, then steering wheel torque applied must make vehicle turn. ASIL=D
AND
SubGoal1: If nominal driving, then there must be sufficient hydraulic flow. ASIL=D
MechSteeringREQ1: If there is sufficient hydraulic flow, then steering wheel torque applied must make vehicle turn. ASIL=D [halted breakdown since pure mechanic]
ASIL decomposition: Primary circuit and SESAMM+ secondary circuit are sufficiently independent
OR
ISO 26262 does not require further breakdown of requirements allocated to non-E/E elements
SubGoal2: If nominal driving, then there must be sufficient hydraulic flow. ASIL=A
PrimaryCircuitREQ1: If nominal driving, then there must be sufficient hydraulic flow. ASIL=C [halted breakdown since pure mechanic]
AND
DriverREQ1: If red warning has been activated for more than or equal to 1 minute, then no nominal driving. ASIL A [halted breakdown since outside of item]
SubGoal3: If no red warning is active or red warning has been activated for less than 1 minute, then during nominal driving, there must be sufficient hydraulic flow. ASIL=A
The motor breaks down after 1 minute.
AND
SESAMMREQ1: If nominal driving and prim. circuit does not provide sufficient hydraulic flow, then red warning is active. ASIL=A
SESAMMREQ2: If nominal driving and prim. circuit has not provided sufficient hydraulic flow for less than one minute, then sec. circuit must provide sufficient hydraulic flow. ASIL=A
SESAMMREQ1.1: If actualEngineSpeed>0 and actualVehicleSpeed > 0 and !actualPrimaryCircuitFlow, then powerSteeringWarning. ASIL A.
SESAMMREQ2.1: If (actualEngineSpeed >0 and actualVehicleSpeed > 0 and !actualPrimaryCircuitFlow) for less than one minute , then actualSecondaryCircuitFlow. ASIL A.
AE421
SecondaryCircuitREQ1: actualSecondaryCircuitFlow equals actuatedSecondaryCircuitFlow. ASIL A. [halted decomposition since pure mechanic]
AE644REQ1: FailureInSteeringCircuit1 must equal powerSteeringWarning. ASIL A.
AE462REQ1: actualEngineSpeed equals EngineSpeed . ASIL A. [halted decomposition since not part of item]
AE31REQ1: actualVehicleSpeed equals WheelBasedVehicleSpeed . ASIL A. [halted decomposition since not part of item]
PrimaryCircuitREQ2: sensedPrimaryCircuitFlow equals actualPrimaryCircuitFlow. ASIL A. [halted decomposition since pure mechanic]
STEE
InfrastructureSWREQ4: actuatedSecondaryCircuitFlow equals RTDB_STEE_ELECTRIC_MOTOR_ACT_E. ASIL A.
InfrastructureSWREQ1: EngineSpeed equals RTDB_ENGINE_SPEED_E. ASIL A.
InfrastructureSWREQ5: sensedPrimaryCircuitFlow equals RTDB_STEE_SENS_CIRC1_LO_E. ASIL A
InfrastructureSWREQ2: WheelBasedVehicleSpeed equals RTDB_VEHICLE_SPEED_E. ASIL A.
InfrastructureSWREQ3: FailureInSteeringCircuit1 equals RTDB_FAIL_STEERING_1_E. ASIL A.
SCANIA
KTH
ESPRESSO

Figure 3.3: *Requirement breakdown of the safety goal of dual circuit steering. From a Scania poster presentation*

3.3.1 Requirement structure FLD

The requirements, building up the requirement structure for the system fuel level display, are shown in Tables 3.1, 3.2, 3.3, 3.4 and 3.5 and are taken from [25].

In the tables, variables are expressed in *italic*. The assumptions, if existing, for every requirement are listed as well as a '*' on assumptions that are fulfilling the requirement.

This notation: FSR_{Driver} means that the requirement is a functional safety requirement, and that it is allocated to the element Driver.

Level I. Safety requirements on CMS and SG

The first safety level is limited to the safety goal of the system and the item (see Definition 2.3), CMS, that is responsible for maintaining the functionality of the fuel level display.

Table 3.1: *Requirements at the highest level in fuel level display*

Name	Component	Assumptions	Description
SG	-	FSR_{CMS}*, FSR_{Driver}	If *actualParkingBrake*[Bool] is not applied (false), THEN *indicatedFuelVolume[%]*, shown by the fuel gauge, is less than or equal to *actualFuelVolume*[%].
FSR_{CMS}	CMS	FSR_{Fuel}*, TSR_{Tank}, FSR_{EMS}, FSR_{ICL}	If *actualParkingBrake*[Bool] is not applied (false), THEN *indicatedFuelVolume[%]*, shown by the fuel gauge, is less than or equal to *actualFuelVolume*[%].

Level II. Safety requirements on the environment of the item (ICL, tank, driver, and EMS)

The second level of requirements are on the environment of the item (CMS), i.e., the assumptions of the requirement FSR_CMS.

Table 3.2: *Requirements at level 2 in fuel level display*

Name	Component	Assumptions	Description
FSR_{Driver}	COO	-	If *actualParkingBrake*[Bool] is not applied (false), THEN the derivative of *actualFuelVolume*[%] is less than or equal to 0.
TSR_{Tank}	Fuel Sensor (T16)	-	The position of the floater *sensedFuelLevel*[%], sensed by the fuel sensor (T16), does not deviate more than 10% from actualFuelVolume[%] in the fuel tank.
FSR_{EMS}	EMS	FSR_{Driver}	If *actualParkingBrake*[Bool] is not applied (false) AND the CAN message *FuelEconomy* is transmitted within 0.3s from the last sent message, THEN the CAN signal *FuelRate[litres=h]* in *FuelEconomy* does not deviate more than 1% from the derivative of *actualFuelVolume[%]*; OR *FuelRate[litres=h]* has the value 0xFE (error).
FSR_{ICL}	ICL	-	If the CAN signal *FuelLevel[%]* in the CAN message *DashDisplay* does not have the value 0xFE (error) and *DashDisplay* is received within 1s from the last received message, THEN *indicatedFuelVolume[%]*, shown by the fuel gauge, is equal to*FuelLevel[%]*. Otherwise, *indicatedFuelVolume[%]* is equal to 0.

Level III. Safety requirements on APPL SW and MIDD SW components

The third level consists of requirements for application software and MIDD software components, that are responsible of providing CAN-signals and sensor readings to the application component as well as encode information from the APP back to CAN-messages.

Table 3.3: Requirements at level 3 in fuel level display

Name	Component	Assumptions	Description
FSR_{Fuel}	fuel	TSR_{Tank}, FSR_{EMS}, FSR_{ICL}, TSR_{ANIN}, TSR^1_{ICAN}, TSR^2_{ICAN}, TSR^1_{OCAN}, TSR^2_{OCAN}	If *actualParkingBrake[Bool]* is not applied (false), THEN *indicatedFuelVolume[%]*, shown by the fuel gauge, is less than or equal to *actualFuelVolume[%].*
TSR_{ANIN}	ANIN	TSR_{ADCC}, $HWSR_{FSens}$	*fuelSensorRes_Val_F32[%]* corresponds to the floater position *sensedFuelLevel[%]*, sensed by the fuel sensor; OR *fuelSensorRes_SS_U08[Enum]* has the value ERR.
TSR^1_{ICAN}	ICAN	SSR_{RCAN}, TSR_{RCAN}	If the CAN signal *FuelRate[l=h]* in the CAN-message *FuelEconomy* does not have the value 0xFE (error) AND *FuelEconomy* has been received within 0.3s from the last received message THEN *fuelRate_Val_F32[l=h]* corresponds to *FuelRate[l=h].*
TSR^2_{ICAN}	ICAN	SSR_{RCAN}, TSR_{RCAN}	If the CAN signal *FuelRate[l=h]* has the value 0xFE (error) OR *FuelEconomy* has not been received within 0.3s from the last received message THEN *fuelRate_SS_U08[Enum]* is set to the value ERR.
TSR^1_{OCAN}	OCAN	TSR_{TCAN}	If *estFuelLevel_SS_U08[Enum]* has the value ERR and the CAN-message *DashDisplay* is sent within 1s from the last sent message, THEN the CAN signal *FuelLevel[%]* in *DashDisplay* has the value 0xFE (error).
TSR^2_{OCAN}	OCAN	TSR_{TCAN}	If *estfuelLevel_SS_U08[Enum]* does not have the value ERR and the CAN-message *DashDisplay* is sent within 1s from the last sent message, THEN the CAN signal *FuelLevel[%]* in *DashDisplay* has a value that corresponds to *estFuelLevel_Val_F32[%].*

Level IV. Safety requirements on BIOS SW components

Requirements on level four is on the BIOS SW components. BIOS-components manages the interaction between software and hardware component e.g. by sending signals corresponding to voltage values to the MIDD-components.

Table 3.4: *Requirements at level 4 in fuel level display*

Name	Component	Assumptions	Description
TSR_{ADCC}	ADCC	$HWSR_{ADC}$, SSR^1_{DMAC}, SSR^2_{DMAC}	*fPinRes_s32[mV]* corresponds to the voltage value *V_Fuel[mV]*.
TSR_{RCAN}	RCAN	$HWSR^1_{CAN}$, SSR^1_{BUFF}, SSR^2_{BUFF}	If *FuelEconomy* is received within 20ms THEN it is available in *FiFoBuffer*.
SSR_{RCAN}	RCAN	SSR^1_{BUFF}, SSR^1_{BUFF}	On *Rcan_getRxMsg_U32()*: If the oldest message in *FiFoBuffer* has PGN 0xFEF2, THEN *rmsg*, corresponding to *FuelEconomy*, is returned.
TSR_{TCAN}	TCAN	$HWSR^2_{CAN}$	On *Tcan_putTxMsg_E(tmsg)*: If *tmsg* has PGN 0xFEFC, THEN *DashDisplay* is eventually transmitted onto CAN.
SSR^1_{DMAC}	DMAC	-	On *Dmac_enableCh(ch U32)*: the DMA channel that corresponds to *ch_U32* is enabled.
SSR^2_{DMAC}	DMAC	-	On *Dmac_disableCh(ch U32)*: the DMA channel that corresponds to *ch_U32* is disabled.
SSR^1_{BUFF}	BUFF	-	On *Buff_put_B(rmsg)*: Adds *rmsg* to *FiFoBuffer*.
SSR^2_{BUFF}	BUFF	-	On *Buff_get_B()*: returns the oldest message *rmsg* from *FiFoBuffer*.

Level V. Safety requirements on COO ECU HW or HW components

The fifth level of safety requirements are requirements on the hardware components allocated to the item, e.g. the fuel sensor.

Table 3.5: *Requirements at level 5 in fuel level display*

Name	Component	Assumptions	Description
$HWSR_{FSens}$	Fuel Sensor (T16)	-	The fuel sensor converts the floater position *sensedFuelLevel*[%] into a voltage value *V_Fuel*[mV] according to table Y3; OR 3000 < *V_Fuel*[mV] OR *V_Fuel*[mV] < 200
$HWSR_{ADC}$	ADC	-	If the DMA channels *timerCh* AND *rfifoCh* are enabled for approx. 20ms, THEN a RAW value of *V_Fuel*[mV] is available in ADCRFIFO.
$HWSR^1_{CANC}$	CAN Controller	-	On *Rcan_decodeCan*: a new CAN message is available in *HWreceivebuffer*
$HWSR^2_{CANC}$	CAN Controller	-	The messages put in *HWsendbuffer* are eventually transmitted onto CAN.

3.3.2 Requirement structure DCS

The dual circuit steering, with the requirement structure in Figure 3.3, is divided into three levels of safety requirements. The highest level resembles the functional level which is requirement on a vehicular level of the system. The second level is on the item level, i.e., the level of the item responsible for using the back-up circuit if the first circuit fails. The thirds level is requirements on the infrastructure software. The requirements are taken from Figure **??**

Level I. SESAMM function level

This subsection has the requirements that is responsible for the functional part of the dual circuit steering and is part of the requirement of the dual circuit steering.

Table 3.6: *Requirements at level 1 in dual circuit steering*

Name	Component	Assumptions	Description
SG	-	SubGoal1*, MechSteeringREQ1	If nominal driving, then steering wheel torque applied must make vehicle turn.
SubGoal1	-	OR{SubGoal2*, PrimaryCircuitREQ1}	If nominal driving, then there must be sufficient hydraulic flow.
MechSteeringREQ1	Mechanical Steering System	-	If there is sufficient hydraulic flow, then steering wheel torque applied must make vehicle turn.
SubGoal2	-	SubGoal3*, DriverREQ1	If nominal driving, then there must be sufficient hydraulic flow.
PrimaryCircuitREQ1	Primary Hydraulic Flow Circuit	-	If nominal driving, then there must be sufficient hydraulic flow.
SubGoal3	-	SESAMMREQ1, SESAMMREQ2	If no red warning is active or red warning has been activated for less than 1 minute, then during nominal driving, there must be sufficient hydraulic flow.
DriverREQ1	Truck driver	-	If no red warning is active or red warning has been activated for less than 1 minute, then during nominal driving, there must be sufficient hydraulic flow.

Level II. AE421 level

Requirements on level two are requirements on the environment on the element AE421 which is the item responsible for the dual circuit steering secondary.

Table 3.7: *Requirements at level 2 in dual circuit steering*

Name	Component	Assumptions	Description
SESAMMREQ1	CMS	AE644REQ1, AE31REQ1, AE462REQ1, PrimaryCircuitREQ2, InfrastructureSWREQ1, InfrastructureSWREQ2, InfrastructureSWREQ3, InfrastructureSWREQ5	If actualEngineSpeed>0 and actualVehicleSpeed > 0 and !actualPrimaryCircuitFlow, then powerSteeringWarning
SESAMMREQ2	Secondary Hydraulic Flow Circuit	AE31REQ1, SecondaryCircuitREQ, AE462REQ1, PrimaryCircuitREQ2, InfrastructureSWREQ1, InfrastructureSWREQ2, InfrastructureSWREQ4, InfrastructureSWREQ5	If actualEngineSpeed>0 and actualVehicleSpeed > 0 and !actualPrimaryCircuitFlow, then powerSteeringWarning
AE644REQ1	ICL	-	FailureInSteeringCircuit1 must equal powerSteeringWarning.
AE31REQ1	COO	-	actualVehicleSpeed equals WheelBasedVehicleSpeed.
SecondaryCircuitREQ1	Secondary Flow Sensor (T56)	-	actualSecondaryCircuitFlow equals actuatedSecondaryCircuitFlow
AE462REQ1	EMS	-	actualEngineSpeed equals EngineSpeed.
PrimaryCircuitREQ2	Primary Flow Sensor (T55)	-	sensedPrimaryCircuitFlow equals actualPrimaryCircuitFlow.

Level III. Infrastructure software level

The infrastructure requirements are requirements on the communication software of the dual circuit steering system.

Table 3.8: Requirements at level 3 in dual circuit steering

Name	Component	Assumptions	Description
InfrastructureSWREQ1	STEE	-	EngineSpeed equals RTDB_ENGINE_SPEED
InfrastructureSWREQ2	STEE	-	WheelBasedVehicleSpeed equals RTDB_VEHICLE_SPEED_E.
InfrastructureSWREQ3	STEE	-	FailureInSteeringCircuit1 equals RTDB_FAIL_STEERING_1_E.
InfrastructureSWREQ4	STEE	-	actuatedSecondaryCircuitFlow equals RTDB_STEE_ELECTRIC_MOTOR - _ACT_E.
InfrastructureSWREQ5	STEE	-	sensedPrimaryCircuitFlow equals RTDB_STEE_SENS_CIRC1_LO_E.

3.4 Database

The requirement data creating the requirement structures is stored in a Neo4J database. Neo4J is a graph database where the data is stored as graphs instead of being stored in tables [16]. In the Espresso project the database data is stored according to a meta model. A meta model is a simplified model of the actual model of the database made for visualizing the structure of the database [9]. The part of the meta model that is relevant for the thesis is shown in Figure 3.4.

The database consists of nodes, which has some information, e.g., a name and a description, etc. The nodes are connected to other nodes with so called arcs, which describes the relationship. A short example describing how the meta model works can be seen in Example 3.1.

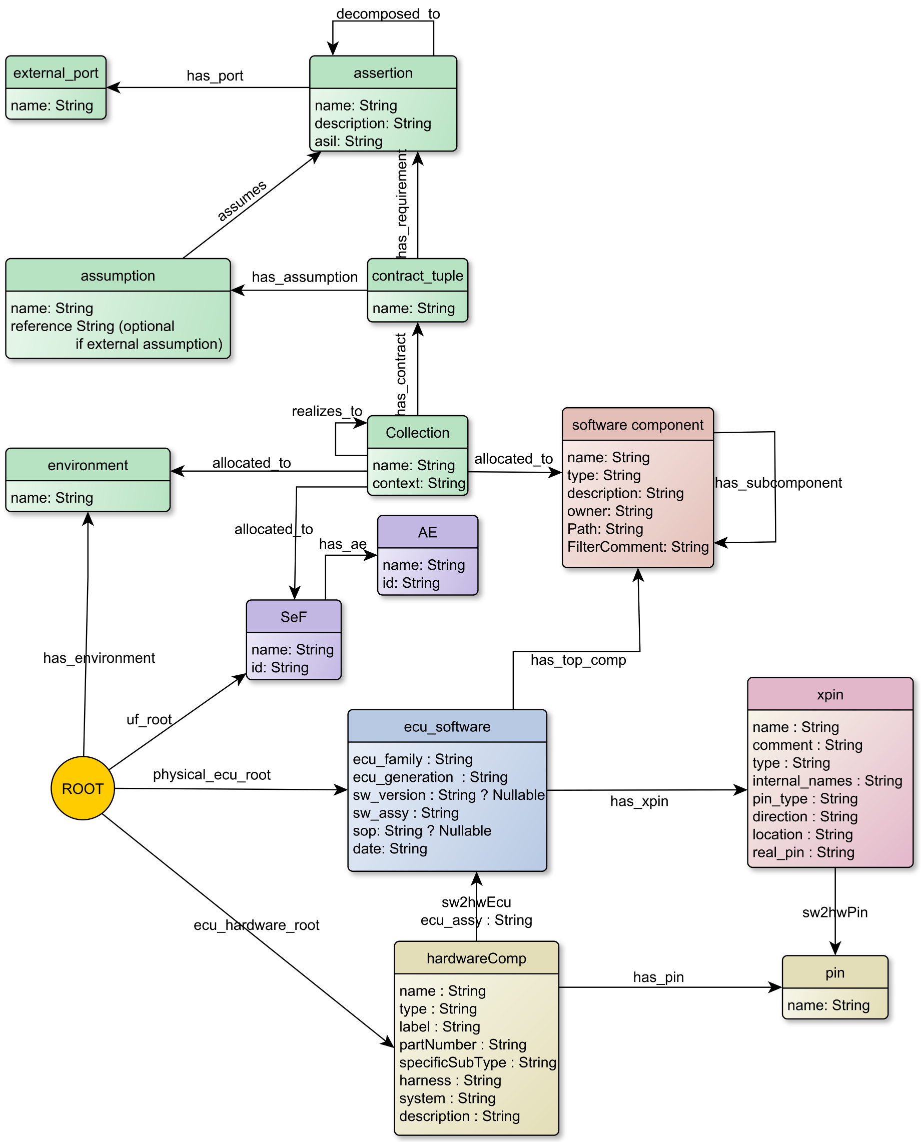

Figure 3.4: *This is a picture of the meta model describing the database structure.*

Example 3.1

This example models the world and people living there, and the results can be seen in Figure 3.5.

The database is consisting of starting node, called root node. The root node has the name *The_World*. *The_World* is connected to the country *Sweden* with the arc called *has_country*. *Sweden* resembles properties in the database that are connected to Sweden.

Sweden is connected with nodes of the type Citizens with the arc *has_citizens*. *Sweden* is also connected to nodes node called *Cities*. *Swedes* has the arcs *lives_in* which connects them with the nodes of the type Cities. It is also connected to itself, which means that a node of the type Citizen can be connected with another Citizen with the arc *has_family*.

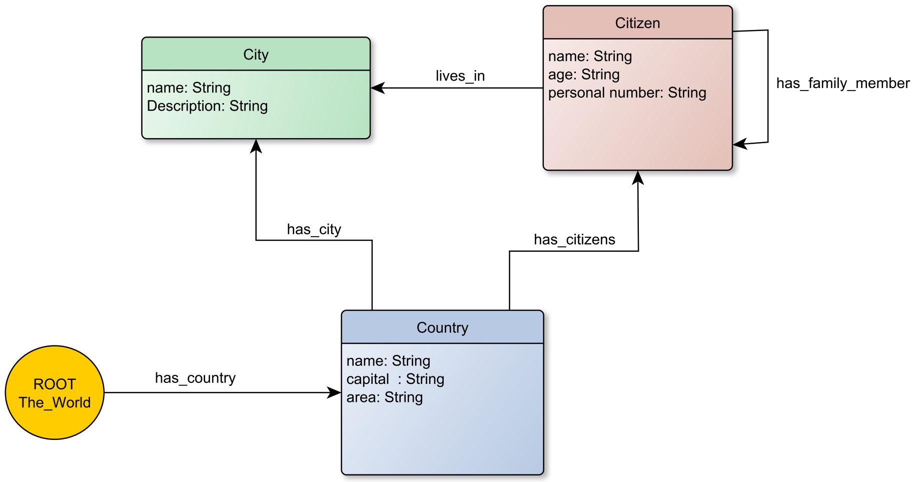

Figure 3.5: *This figure describes a test database with a few node types connected with arcs describing their relationships.*

The meta model that describes the database used in this master thesis can be seen in the Figure 3.4. It describes the relationship between, e.g., how software components are connected to a top component as sub components, which resembles an abstraction layer on an ECU, that in turn is connected to an ECU software.

The main focus is on the nodes in the top of the figure called contract_tuple, assumption, assertion and collection. These are the types that build the requirement structures that are used in the fault tree generation.

A collection consists of a group of contracts that are all allocated to the same component. A contract consists of, as defined in Definition 2.15, a guarantee, which is the assertion that it is connected to with the arc *has_requirement*, as well as assumptions connected with the arc *has_assumption*. The information of the contracts allocated component comes from the collection it belongs to, since all collections are connected to one component.

The assertion type consists of all requirements in the database. Assertions have an arc to itself which says *decomposed_to*. Decomposition means that an assertion is broken down into another assertion which fulfills the first one. Fulfills, in this case, means that the fulfilling assertion overtakes the responsibility that the first assertion is fulfilled, i.e., as defined in Definition 2.2.2. The assumption type is either a reference to an assertion (see above) or an external assumption, which is an assumption to a contract, see Section 2.2.

4 Fault tree generation

This chapter describes the fault tree generation method together with the preliminaries that are needed to create the trees.

The motivation to why fault trees can be created from requirements at all is that requirements in a way is the inverse to a fault tree and can be compared to a success tree which is mentioned in [20]. A success tree describes which basic events has to work to guarantee that the top event does **not** occur. In the similar way a requirement structure is built up on which part of a system that has to work to be able to guarantee that the safety goal of the system is fulfilled.

The concept of turning a success tree into a fault tree is an easy procedure and can be done by changing the top event from not occurring into occurring and turning all OR-gates into AND, and the other way around [20]. The requirement structure is more complex but the general idea is the same.

A contract promises that a scenario fulfilling the assumptions will result in a scenario defined by the guarantee. This means that it is possible to create a success like tree from the requirement structure. However when it comes to a scenario not fulfilling the assertion of the assumptions it is only possible to say that the guarantee is not fulfilled. Therefore, it can be useful to use failure modes to try to describe how broken assumptions will affect the guarantee, i.e., how failure states in the assumptions maps on to a failure state in the guarantee. This is illustrated in Figure 4.1

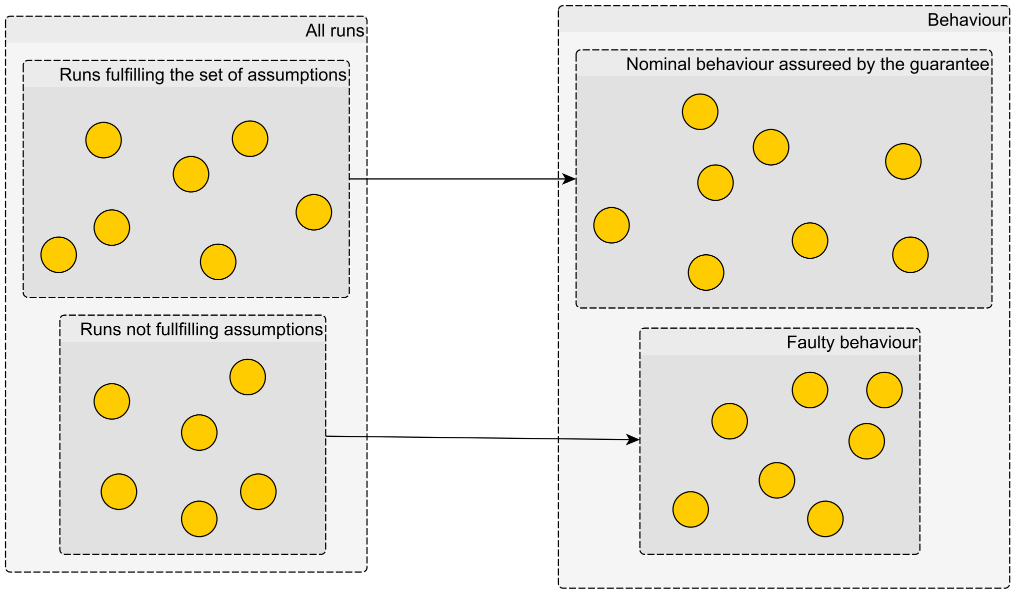
All runs
Runs fulfilling the set of assumptions
Runs not fullfilling assumptions
Behaviour
Nominal behaviour assureed by the guarantee
Faulty behaviour

Figure 4.1: *The mapping from fulfilled assumptions to nominal behaviour and from unfulfilled assumptions to faults.*

4.1 Premises

The extension of the requirement structure with failure propagation will make it possible for a non-fulfilled assumption not to violate the safety goal. This can be done if the failure mode that propagates does not cause any hazards.

The premise of the thesis is to generate fault trees based on the information gathered in a requirement structure. Requirement nodes, in the requirement structure, is made up of contracts, see Definition 2.15. For the guarantee of the contract to hold, the assumptions and the element responsible for the behavior has to be fulfilled. This means that in the general case of contract theory, to ensure that the safety goal is fulfilled, all the assumptions and components in the contract structure have to be fulfilled, according to Definition 2.9. However, the relation between hazards and safety goals are many to many, i.e., violating a safety goal in different ways can lead to several hazards and the occurrence of a hazard can lead to several violated safety goals. Therefore it is not enough to know that requirements (especially safety goals) are broken, it is also necessary to know **how** the requirements are broken. Example 4.1 explains how different failure modes violating requirements will result in different scenarios.

Example 4.1

The requirement FSR_{Fuel} from the fuel level display system have a few assumptions that is required to be fulfilled. Depending on what failure modes the assumptions has the systems will be affected in different ways.

Consider failure of the assumption TSR_{Tank} from Table 3.2. The position of the floater *sensedFuelLevel*[%], sensed by the fuel sensor, does not deviate more than 10% from actualFuelVolume[%] in the fuel tank.

This requirement is broken if the floater in the sensor shows more than 10% more fuel than it exists in the tank. This situation can cause the hazard OutOfFuel since the driver will believe there is more fuel than it actually is. Instead, if the fuel sensor instead show 10% less fuel the driver will have to refuel a truck earlier than it needs to. Both of these scenarios will break the requirement TSR_{Tank}, which will cause a violation of the safety goal, but only one of the faulty behaviours will cause the OutOfFuel hazard.

To be able to identify the correct hazard from the requirement structure, failure modes must to be defined. This will make it possible to identify the failure modes of the items when causing the hazard. e.g., the floater in the tank can cause OutOfFuel if it has the failure mode *signal_value_high* but not if it has *signal_value_low*.

4.2 Failure mode propagation network

As a conclusion from Section 4.1, failure modes are needed to correctly create a fault tree from the requirement structure. The failure modes will be added to all the nodes in the requirement structure, i.e., all requirements and items.

The new network should contain all requirements and elements from a requirement structure as well as hazards connected to the safety goal. The hazards are connected to certain failure modes that exist in the safety goal. For the network to be useful, it is necessary to describe the failure mode propagation from a fault in an item to the requirement(s) on the item, and from the requirement(s) through the whole requirement structure.

From the network it should be possible to see which failure mode in a specific element that can cause a selected hazard. A list of criteria on the new failure mode propagation network is listed below.

Required properties of the failure propagation network:

1: Ability to, for each node, describe different states (failure modes).

2: Describe how states in nodes affects the states of their adjacent nodes.

3: Traceability of top level state to identify what causes it.

A property of the Bayesian network is that, because of the CPT handling all the state propagation, it is not needed to create more nodes than the requirements and components that comes from the requirement structure. That property makes the Bayesian network easy to overview. A big disadvantage is that the Bayesian network CPT will grow exponentially with the number of failure modes and parents, which may cause problems when dealing with nodes with a lot of parents.

Pros:

1: It can describe all the states, i.e. failure modes, that each node can be in.

2: It is possible to describe all possible propagation scenarios using the conditional probability tables.

3: It is possible to, using inference, see which components that can affect a hazard.

4: Well used method with a lot of already existing software e.g. SMILE [13]

5: Can be used to generate FMEA if starting from component and investigating which hazards that are affected.

6: Possibility to extend so the failure propagation network contains true probabilities, rather than if/if not components/requirements are affected.

A Bayesian network can describe all possible states that it could be in, given some evidence, at the same time it can describe all possible hazardous events that can occur given that an element has a certain failure mode, like a FMEA. It can also identify which component failure modes that could cause a hazard, i.e., a fault tree.

The fact that Bayesian networks are directed acyclic graphs will not be a problem since the requirement structure, that the failure propagation network is based on, also is directed and acyclic, see Definition 2.2.2.

Cons:

1: State explosion in the CPT when there exist many failure modes as well as many parents to a child.

2: It is a bit inefficient when many of the rows in the CPT wont be used, and/or can be defined in an easier way.

A problem with Bayesian networks in this case is that parts of the CPT will not be utilized, e.g., the usage of probability will be seen more like a binary state, either something propagates or it does not. Also it is likely that some combinations of failure modes in the parents never will occur which will make parts of the CPT unnecessary. A problem occurs when there are many failure modes as well as parents to a node because of the number of elements in the CPT will be large according to (2.7)

An already existing tool for safety analysis in ISO26262 is Hierarchically Performed Hazard Origin & Propagation Studies (HiP-HOPS). Hip-HOPS also has the possibility to automatically synthesize fault trees and FMEA. Hip-HOPS uses models created by external modeling programs compatible with Hip-HOPS, e.g., Matlab Simulink or SimulationX to generate their fault trees and FMEA [18]. Since the data in this thesis is in the shape of requirements, HiP-HOPS can not be used in this thesis.

4.2.1 Defining failure mode propagation

Since a Bayesian network contains more information than it is needed of the failure mode propagation network‘, logical statements will be used to define the failure mode propagation. The statements will later be translated into conditional probability tables. This method will make it easier to define how parents failure modes propagate to the child nodes.

This method can be used since it is likely that many of the columns in the CPT can be expressed using a basic logical expression. Also it is cumbersome as well as error prone to generate conditional probability tables by hand.

The conversion to CPT is done for each row separately. Each row is described by a, maybe empty, set of equations, A. A is connecting a failure mode of the node to failure modes of the parent nodes. The union of all the equations E_i in A, will give a new composite equation E*. E* takes the parents failure modes as argument and checks if the value that is returned is 0 or not. If the return value is $\neq 0$ then the number 1 will be added to the tested position in the table. This will resemble that there is a probability > 0 of propagating to that failure mode given that combination of failure modes of the parents.

The given procedure will be done for all rows until each element has been processed. The last step is to normalize each column so it will sum to 1. The method will be shown in Example 4.2.

Example 4.2

A requirement node, R_0, has two failure modes FM_1, FM_2, and the no fault mode NF. The requirement has two parent nodes, R_1 and R_2, with the same failure modes. The CPT of requirement R_0 will have $3^{2+1} = 27$ elements according to Equation (2.7).

Consider that there exist the following relationships between the failure modes in R_0 and in R_1 and R_2:

$$\begin{aligned}
&R_0(NF) = R_1(NF) \wedge R_2(NF) \\
&R_0(FM_1) = R_1(FM_1) \vee R_2(FM_1) \vee \{R_1(FM_2) \wedge R_2(FM_2)\} \\
&R_0(FM_2) = R_1(FM_2) \vee R_2(FM_2)
\end{aligned}$$

then, by automatically transforming the results above, the resulting CPT can be seen in Table 4.1.

Starting with the first row R(NF) which has the equation $R_0(NF) = R_1(NF) \wedge R_2(NF)$. The first column has failure modes $R_1(NF)$ and $R_2(NF)$ which will get the result 1 since $1 \wedge 1 = 1$. All other rows will be 0, since $1 \wedge 0 = 1$, $0 \wedge 1 = 0$, $0 \wedge 0 = 0$.

Next row corresponds to equation $R_1(FM_1) \vee R_2(FM_1) \vee \{R_1(FM_2) \wedge R_2(FM_2)\}$. The expressions are calculated for every column and their values are set to the specific position in the table.

The equation $R_1(FM_2) \vee R_2(FM_2)$ is used for the last row, with the same principle. When all elements in the table has been processed all the columns are normalized so they sum to 1, i.e., by calculating the sum of each column and dividing each element in the column by the sum.

Table 4.1: *Results of the conditional probability table generated by the equations describing the propagation*

R_2	NF			FM_1			FM_2		
R \ R_1	NF	FM_1	FM_2	NF	FM_1	FM_2	NF	FM_1	FM_2
NF	1	0	0	0	0	0	0	0	0
FM_1	0	1	0	1	1	0.5	0	0.5	0.5
FM_2	0	0	1	0	0	0.5	1	0.5	0.5

Unless the complexity of the failure mode is very high, this method will simplify the generation of failure mode propagation compared to manually generating conditional probability tables.

Since each row can be described using one, possibly complex, equation, the number of equations describing the CPT will be equivalent of the number of failure modes in the node. For most cases, simple logical equations can be found that will describe the propagation, which will lower the manual labour when generating the CPTs.

4.3 Bayesian network generation

The following section describes how the Bayesian failure propagation network is generated and how it can be used to find out which hazards can be caused by different component faults.

4.3.1 Generating Bayesian networks

The Bayesian networks will be generated using the software libraries [13]. First the requirement structure will be built as a Bayesian network starting from a safety goal, and then by using the meta model in Section 3.4 all the connected requirements will be retrieved. Also the allocated elements will be retrieved.

All retrieved requirements will be introduced into a DAG as nodes connected to other requirements and elements with arcs. Also hazard nodes will be connected to the safety goal as well as introducing failure modes in each element and requirement. A method for adding failure modes to all nodes will be done as described in Section 4.2. By adding probabilities of failure mode propagation and for each node putting them in a conditional probability table the DAG is made into a Bayesian network.

First the failure modes have to be defined in every requirement and elements. The Bayesian network now contains information about the structure of the system and which failure modes each element and requirement has. Information on how the failure mode propagation is taken from the failure mode propagation described in Section 4.2.1 above.

Finding causes to a hazard is done by inducing a hazard and comparing it to the nominal case. A hazard is induced by setting P(Hazard = Active) = 100% and updating the conditional probabilities in the network according to Equation (2.1), and then checking $P(N_i = nominal|Hazard = Active) = p_i$, where N_i is a node in the system and p_i is a probability. The possible causes is following set nodes $\{N_j|P(N_j = nominal|Hazard = active) > 0, N_j \in allnodes\}$.

Using this, by first looking at the nominal case, i.e. setting the probability of all hazards to 0, the probability of being in the nominal state is 100%. Then a hazard that, the fault tree will be created from, will be induced. The hazard will be induced by setting the probability of the hazard to occur to 1. It is now possible to see how the conditional probabilities changes and then comparing the probability of being in the nominal state for each component and requirement compared to the state where the probability of there existing hazards being 1.

All the components and requirements that in some way are affecting the hazard will be used in a new Bayesian network which only contains possible causes to the hazard.

4.4 Automated fault tree generation

This section describes the automated fault tree generation process and how the results can be used for fault tree analysis.

4.4.1 Generating fault trees

From the generated Bayesian network the creation of fault trees start with choosing a safety goal, i.e., what part of the Bayesian network that will be used. The next step is choosing for which hazard to create the fault tree from.

The first step is to find out which requirements and components will affect the specified hazard. This can be done in various ways but the chosen method is to calculate the probability of nominal behaviour for requirements and components, given a hazardous state and the comparing it to the nominal state. This method will single out the requirements and components that in some way can cause the hazard.

A new network of nodes only consisting of components and requirements, that has a probability of causing the hazard, will be created.

Each component node will be a basic node in the fault tree since they are the atomic part of what can result in the safety goal being violated and in that way causing a hazard. If a component is allocated to more than one requirement the component will be cloned and one component node will being allocated to each requirement, this action is necessary to create a tree structure out of the Bayesian DAG. If a requirement R is the assumption of more than one requirement, R will be copied together with its child nodes and then one copy will be allocated to each of the parent requirements. A downside to this is that it is no longer possible to see that the requirements are the same anymore.

When the directed acyclic graph is turned into a tree, the requirement nodes are turned into logical AND/OR-gates. If there exist redundancy the node in the Bayesian network will have an identifier that says it is an AND-node. The Bayesian network has the support to define if it is an AND-node in the CPT:s. An OR-node in the Bayesian network can be seen in the CPT if failure mode in all of the parents alone cause a failure mode in the requirement node. If it is needed for two or more nodes in all situations for failure mode propagation to occur, there is an AND-node between them.

If redundancy exist, it is possible to describe the redundancy as as an additional node, existing in the Bayesian network. This node resembles the OR-node used in the extended contract theory in Definition 2.20. Another way of describing redundancy is to use the conditional probability tables to describe that two or more elements are redundant.

The tree is created from the new treelike Bayesian network where all the requirements are defined as logical gates. The tree is generated from the top, i.e. from the hazard. First logical nodes are created depending on the requirement nodes in the new Bayesian network. Then, components are added as basic events in every step. Before the components are added, the tree is analyzed to see if basic event already have been added higher up in the tree. If the basic node already exists, the new node will not be added. Since the limitations of the thesis assumes that events are independent, adding the same event will be redundant and not change the information of the tree. The proof of this can be seen in Theorem 4.3. In reality the events is not necessarily independent which will cause a loss of information.

Theorem 4.3. *A basic event A will be redundant if a basic event A', where A = A', exists higher up in the tree structure.*

Proof: Proof of Theorem 4.3 using mathematical induction
The basic events representing the component comp are called A and A' (to separate them), where A = A'. The hierarchy level of LVL(A) $\leq$ LVL(A'). For each logical gate further down in the tree the level will increase by 10, i.e., the higher level the lower in the tree hierarchy. An illustration of this can be seen in Figure 4.2

$\circ$ and $\diamond$ denotes arbitrary logical gates (AND/OR). Using following expressions:

$$\text{Idempotence of } \wedge \qquad a \wedge a = a \tag{4.1}$$

$$\text{Idempotence of } \vee \qquad a \vee a = a \tag{4.2}$$

It is possible to show redundancy occurs if two components are added to the same level in the tree structure.

$$A \circ A' \circ B \circ C \circ \cdots \overset{(4.1),(4.2)}{=} A \circ B \circ C \circ \ldots \tag{4.3}$$

Redundancy when two identical basic events occur on the same level are proven. Next step is to prove that it still holds if LVL(A') - LVL(A) = 1.

$$A \circ B \circ C \circ \cdots = [B = (A' \diamond B' \diamond C' \diamond \ldots)] = A \circ (A' \diamond B' \diamond C' \diamond \ldots) \circ C \circ \ldots \tag{4.4}$$

There are two cases to take into account, $\circ = \diamond$ and $\circ \neq \diamond$. And to solve them following relations are used:

$$\text{Associativity for } \vee \qquad a \vee (b \vee c) = (a \vee b) \vee c \tag{4.5}$$

$$\text{Associativity of } \wedge \qquad a \wedge (b \wedge c) = (a \wedge b) \wedge c \tag{4.6}$$

$$\text{Absorption 1} \qquad a \wedge (a \vee b) = a \tag{4.7}$$

$$\text{Absorption 2} \qquad a \vee (a \wedge b) = a \tag{4.8}$$

$$\text{Distributivity of } \wedge \text{ over } \vee \qquad a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) \tag{4.9}$$

$$\text{Distributivity of } \vee \text{ over } \wedge \qquad a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \tag{4.10}$$

Case: $\circ = \diamond$

$$A \circ (A' \diamond B' \diamond C' \diamond \ldots) \circ C \circ \cdots \overset{(4.5),(4.2)\ \mathit{or}\ (4.6),(4.1)}{=} A \circ (B' \diamond C' \diamond \ldots) \circ C \circ \ldots \tag{4.11}$$

Case: $\circ \neq \diamond$

$$A \circ (A' \diamond B' \diamond C' \diamond \ldots) \circ C \circ \cdots \overset{(4.7)\ \mathit{or}\ (4.8)}{=} A \circ C \circ \ldots \tag{4.12}$$

Now LVL(A') - LVL(A) = 2. Depending on if gate at LVL(A) and LVL(A) + 1 are the same or not the associativity law ((4.5) or (4.6)) or the distributive law((4.9) or (4.10)) will be used. Now the results in (4.11) or (4.12) can be used to prove redundancy when the level difference is 2.

If LVL(A') - LVL(A) = n and n $\geq$0, associativity law or distributive law is used depending on gate types between LVL(A) and LVL(A) + 1. The results from LVL(A') - LVL(A) = n - 1 can then be used prove that redundancy will occur when the level difference is n, where n is arbitrary.

This concludes the proof that all recurrences of component on a higher level in the fault trees are redundant. $\square$

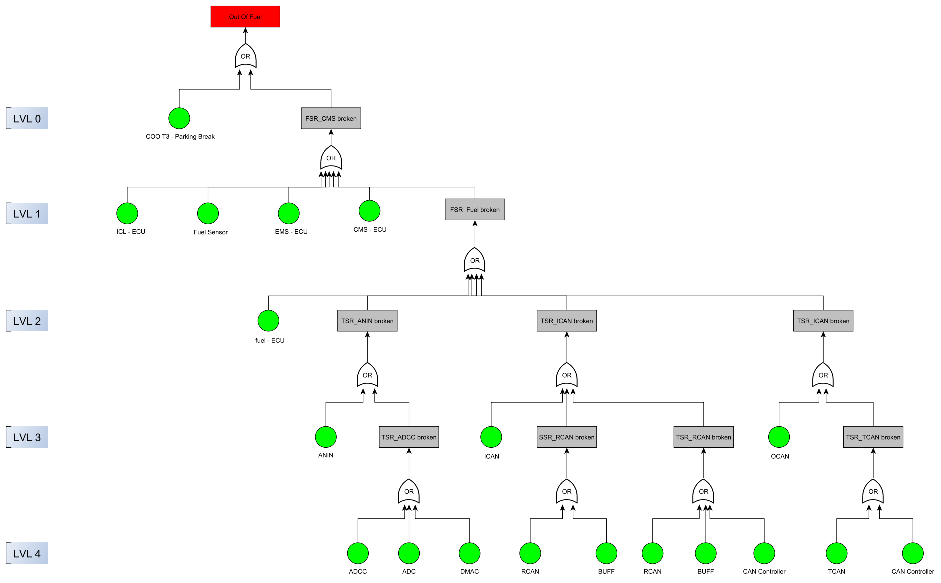

Figure 4.2: *Here is an illustration of the concept levels in a fault tree. The first level, lvl 0, is defined as the events beneath the logic gate connected to the hazard.*

The basic events are created from the component nodes that are allocated to each requirement node. Before a new basic event is added, the tree is first checked to see if that basic event exist in a higher hierarchy in the tree (closer to the hazard). If the component already exist the node is not added

The fault tree is finished with creating intermediate event over each of the nodes in the fault tree. The intermediate nodes is describing what requirement that has been broken if the result from the gate is true.

The last step in the generation is to perform fault tree analysis on the system. The analysis is done by calculating the minimal cut sets, which are the minimal causes of the hazard, of the system. The minimal cut sets from the tree is calculated using a method described in the flow chart in Figure 4.3. The first part in calculating the minimal cut set is to recursively go through all the branches of the tree, starting from the root node (the hazard). When reaching leaf nodes the method goes upwards toward the root node. In each junction (gate) a logical expression, describing the possible failure causes, is calculated. The expression is calculated by using the gates logical operator (AND/OR) on all incoming branches. If the gate is an OR-gate the idempotent law for OR and AND is used, giving an expression on the following form:

$$\text{Exp1 OR Exp2 OR Exp3 OR} \ldots \tag{4.13}$$

For AND-nodes the distributive law is first used, followed by the idempotent law giving, the form seen in Equation (4.13). Next all expressions are compared and all super-sets will be removed. This calculations will be done in each gate until the root node has been reached.

The minimal cut set can then be used to generate a minimal tree describing only the minimal combination of basic events causing the hazard.

4.5 Meta model extension

The meta model that describes the database, shown in Figure 4.4, has to be extended to contain failure modes and hazards. These extensions are necessary to store and retrieve information that is needed to create fault propagating Bayesian networks. The failure mode type contains a name and description as well as the relationship *has_failure_mode* with the assertion type. Also software and hardware has the relationship *has_failure_mode* with the failure mode type. The hazard type is connected failure modes as well as assertions with the relationship called *caused_by* which says that a hazard will be caused by an assertion being in a certain failure mode.

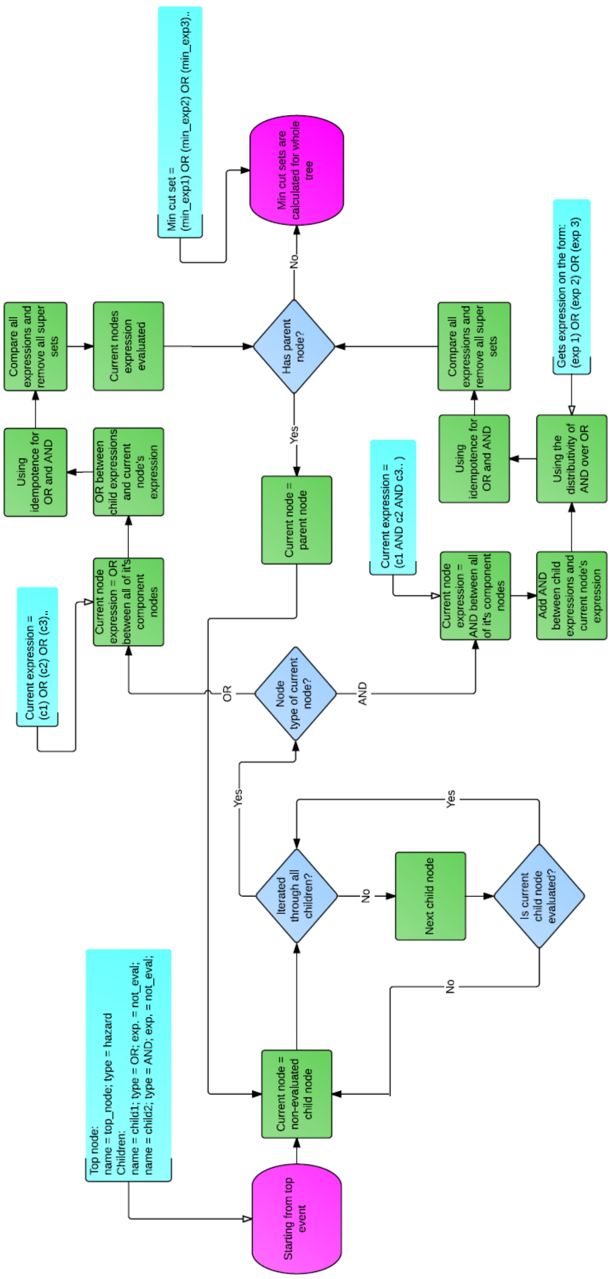
Staring from top event
Top node:
name = top_node; type = hazard
Children:
name = child1; type = OR; exp. = not_eval;
name = child2; type = AND; exp. = not_eval;
Current node = non-evaluated child node
Iterated through all children?
No
Next child node
Is current child node evaluated?
Yes
No
Yes
Node type of current node?
OR
AND
Current node expression = OR between all of it's component nodes
Current expression = (c1) OR (c2) OR (c3)..
OR between child expressions and current node's expression
Using idempotence for OR and AND
Compare all expressions and remove all super sets
Current nodes expression evaluated
Current node expression = AND between all of it's component nodes
Current expression = (c1 AND c2 AND c3..)
Add AND between child expressions and current node's expression
Using the distributivity of AND over OR
Gets expression on the form: (exp 1) OR (exp 2) OR (exp 3)
Using idempotence for OR and AND
Compare all expressions and remove all super sets
Has parent node?
Yes
Current node = parent node
No
Min cut sets are calculated for whole tree
Min cut set = (min_exp1) OR (min_exp2) OR (min_exp3)..

Figure 4.3: *A flowchart for the algorithm that calculates minimal cut sets from the generated fault tree.*

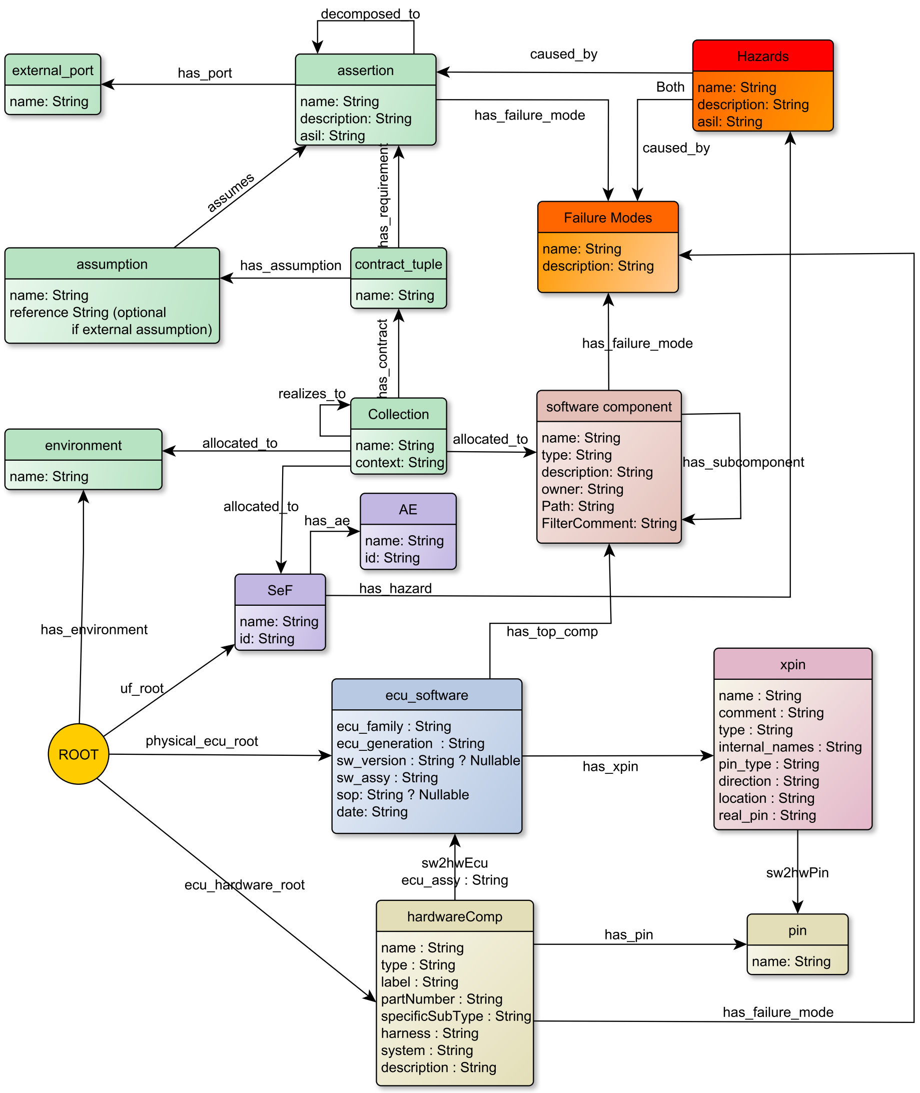

Figure 4.4: *This figure is the extended meta model which describes the new database structure.*

5 Case study

Since there are no available fault trees for the case study systems, the validation of the method is done in two steps. The first step is to manually generate fault trees from to requirement structure to validate that the generated trees are correct with regard to the system described by the requirement structure.

The second step is to evaluate the possibility to generate fault trees of systems starting from requirement structures.

5.1 Results evaluation: Fuel level display

The two case study examples have been automatically generated in accordance to the method described in Chapter 4 together with requirement structures of the both systems

5.1.1 Fuel level display manual fault tree generation

The fault tree generated for fuel level display is generated from the Hazard **OutOfFuel** shown in Figure 6.5. The generation of this hazard is made from first creating a failure propagation network from the requirement structure of fuel level display. The hazard connected to the system is chosen to create the fault tree from. The fuel level display has two hazards, **OutOfFuel**, shown in Figure 6.5, and **NoFuelLevelWarning**. **OutOfFuel** is the hazard that the truck runs out of fuel when driving.

The hazard **OutOfFuel** can be caused if the safety goal has the failure mode *signal_value_high*. This failure mode says that the requirements regarding the amount of fuel in the fuel tank is higher than the actual amount. The information of the failure modes as well as hazards, in fuel level display, comes from a system FMEA of system [12].

The relevant failure modes in this example is the failure modes that can cause the safety goal to be in the *signal_value_high* failure mode. The failure mode *signal_value_high* implies that a too high signal exist in the item. The other important failure mode is the failure mode *signal_value_omission* which means that no signal has been transferred.

In this case study it is assumed that all requirements has the following failure modes: *signal_value_high*, *signal_value_low*, *signal_value_omission*, *signal_value_error* and *signal_value_commission*. It is assumed that the failure mode *signal_value_high* propagates into the same failure mode, i.e. *signal_value_high*. Also the failure mode *signal_value_omission* is assumed to be able to cause the *signal_value_high* failure mode, since not delivering signals will cause the system not to update the fuel level.

The failure mode propagation in the network is defined as in Table 5.1. In the table only the failure propagation of the relevant failure modes, i.e. the ones that cause the OutOfFuel hazard as well as the no fault failure mode, are presented. The table shows a general propagation where a requirement or item has both the failure modes signal_value_omission and signal_value_high and how failure modes in the parent could cause them.

Table 5.1: *The failure mode propagation in FLD, with the parents in {parent}*

Failure Mode	**Parent Failure Mode Combinations**
NF	{1}NF AND {2}NF AND.. AND {n}NF
signal_value_high	({1}signal_value_high OR {1}signal_value_omission) OR ({2}signal_value_high OR {2}signal_value_omission) OR ⋮ OR ({n}signal_value_high OR {n}signal_value_omission)
signal_value_omission	{1}signal_value_omission OR {2}signal_value_omission OR ⋮ OR {n}signal_value_omission

When failure modes as well as failure mode propagation are added to all requirements and elements it is possible to evaluate which requirements and elements are affecting the hazard OutOfFuel using the method described earlier. The results show that all the elements and requirements, in the requirement structure, are possible causes to the hazard.

In the next coming sections the requirements describing fuel level display will be gone through and used to manually create a fault tree. The manually created tree will have the hazard OutOfFuel as top event. The manually generated tree will later be compared to the results from the fault tree generation.

When generating fault trees, the error handling is not considered, and to make a more accurate comparison the error handling will not be considered when manually generating the fault tree.

Level 1. Safety requirements on CMS and SG

The requirements from the first safety level is taken from Table 3.1 considering the safety goal of the system as well as the item, CMS.

The hazard **OutOfFuel** is connected to the safety goal and occurs when the *indicated-FuelVolume[%]* is higher then *actualFuelVolume*[%]. The hazard will be the top event in the fault tree. SG is fulfilled by the requirement FSR_{CMS} as well as having an assumption FSR_{Driver}, which will be described as the contract $(\{FSR_{CMS}, FSR_{Driver}\}$, SG).

According to Definition 2.15 the contract is fulfilled if the allocated element as well as the assumptions are fulfilled. Since the goal is on the vehicle level it has not any element allocated which means it will be violated if any of the assumptions are broken. Since there is no additional error handling this means that the safety goal in the fault tree will be turned into an OR-gate with the the inputs FSR_{CMS} and FSR_{Driver}.

The fulfilling requirement FSR_{CMS} is violated if its element, the ECU system CMS (Chassis Management System) is faulty or the assumptions FSR_{Fuel}, TSR_{Tank}, FSR_{EMS} and FSR_{ICL} are violated. There is no error handling on this stage either which means that FSR_{CMS} will be turned into an OR-gate with the basic event CMS and the undeveloped events FSR_{Fuel}, TSR_{Tank}, FSR_{EMS} and FSR_{ICL}.

Since there are no more information about the assumptions at level 1, FSR_{Driver} will also be a undeveloped event in the fault tree. Figure 5.1 describes the fault tree of the first level of the requirement structure of the fuel level display.

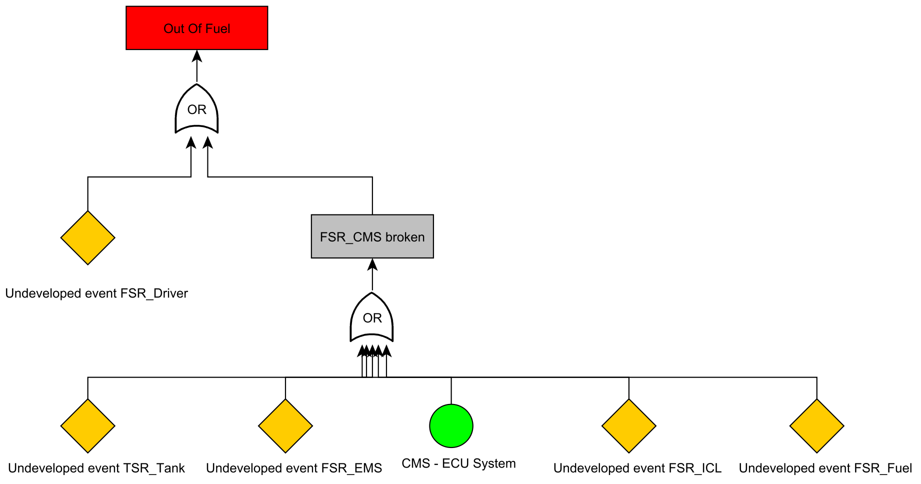
Out Of Fuel
OR
Undeveloped event FSR_Driver
FSR_CMS broken
OR
Undeveloped event TSR_Tank
Undeveloped event FSR_EMS
CMS - ECU System
Undeveloped event FSR_ICL
Undeveloped event FSR_Fuel

Figure 5.1: *A fault tree describing the first level of fuel level displays safety requirement*

Level 2. Safety requirements on the environment of the item (ICL, tank, driver, and EMS)

The next level of requirements considers the environment of the item (CMS) which are the assumptions of FSR_{CMS}. The requirements from level 2 of the fuel level display comes from Table 3.2.

The assumption of the safety goal, FSR_{Driver}, is allocated to the element Driver does not have any assumptions. FSR_{Driver} will be turned into the basic event Driver in the fault tree.

TSR_{Tank} does not have any assumptions which makes the allocated element, the fuel tank, the basic event. The FSR_{EMS} assumption FSR_{Driver}, since the basic event Driver exist higher up in the tree it according to Theorem 4.3 be redundant if added lower in the tree. The requirement will instead be turned into the basic event of the element, the EMS ECU System. The FSR_{ICL} does not have any assumptions either which makes the element, the ICL ECU system, a basic event.

At level 2 there is no information about the fuel filter FSR_{Fuel} and it will be added to the fault tree as an undeveloped event, which means that there is not enough information about that part yet. The fault tree describing the second safety requirement level can be seen in figure 5.2.

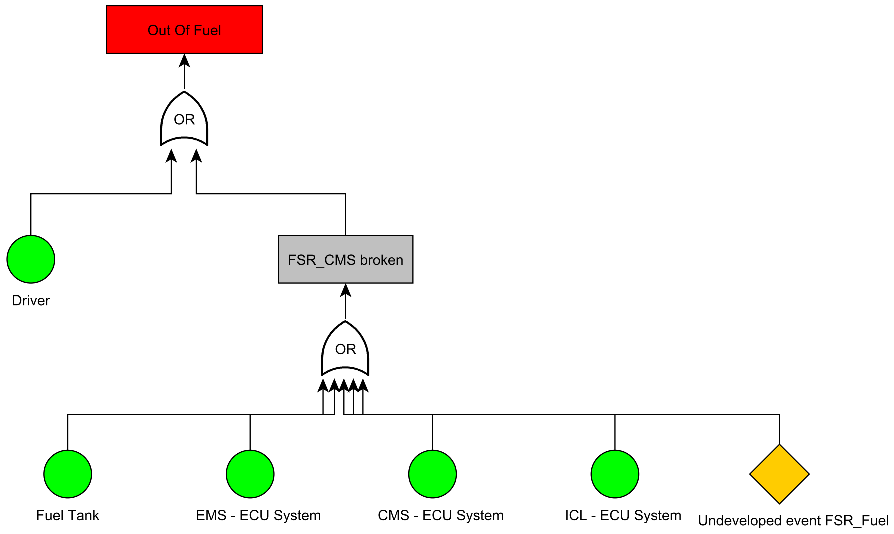

Figure 5.2: *A fault tree describing the first two levels of FLDs safety requirements*

Level 3. Safety requirements on APPL SW and MIDD SW components

The third level consists of application software components, and MIDD software components that are responsible of providing CAN-signals and sensor readings to the application component as well as encode information from the APP back to CAN-messages. The data comes from Table 3.3.

FSR_{Fuel} is fulfilling the requirement FSR_{CMS} and has the assumptions TSR_{Tank}, FSR_{EMS}, FSR_{ICL}, TSR_{ANIN}, TSR^{1}_{ICAN}, TSR^{2}_{ICAN}, TSR^{1}_{OCAN} and TSR^{2}_{OCAN} and is allocated to the element Fuel, which is the filter calculating the fuel level. The requirement has no error handling and will therefore represented as an OR-gate with inputs from all the assumptions. Since TSR_{Tank}, FSR_{EMS}, FSR_{ICL} exist higher up in the tree they will not be added. It leaves the requirements TSR_{ANIN}, TSR^{1-2}_{ICAN} TSR^{1-2}_{OCAN} as inputs.

The requirement TSR_{ANIN} is needed to guarantee that the correct value from the fuel sensor has been received, to do so the element ANIN as well as the assumptions TSR_{ADCC} and $HWSR_{FSens}$ has to be fulfilled. TSR_{ANIN} will be an OR-gate with 3 inputs, 2 from the assumptions and 1 from the basic event ANIN.

TSR^{1-2}_{ICAN} both are depending on SSR_{RCAN} and TSR_{RCAN} as well as the element ICAN to be fulfilled. Since the two requirements are identical when it comes to the allocation element and assumptions it is enough to represent one of them in the fault tree. The requirement will be turned into an OR-gate with 3 inputs, two from the assumptions and one from the element ICAN.

The same procedure can be done with TSR_{OCAN}^{1-2} and is in the same way an OR-gate with two inputs, one from the element OCAN and one from the assumption TSR_{TCAN}.

The events corresponding to the requirements TSR_{ANIN}, TSR_{ICAN} and TSR_{OCAN} are the only events that are not developed in this step.

Figure 5.3 resembles the fault tree described by the first three requirements levels.

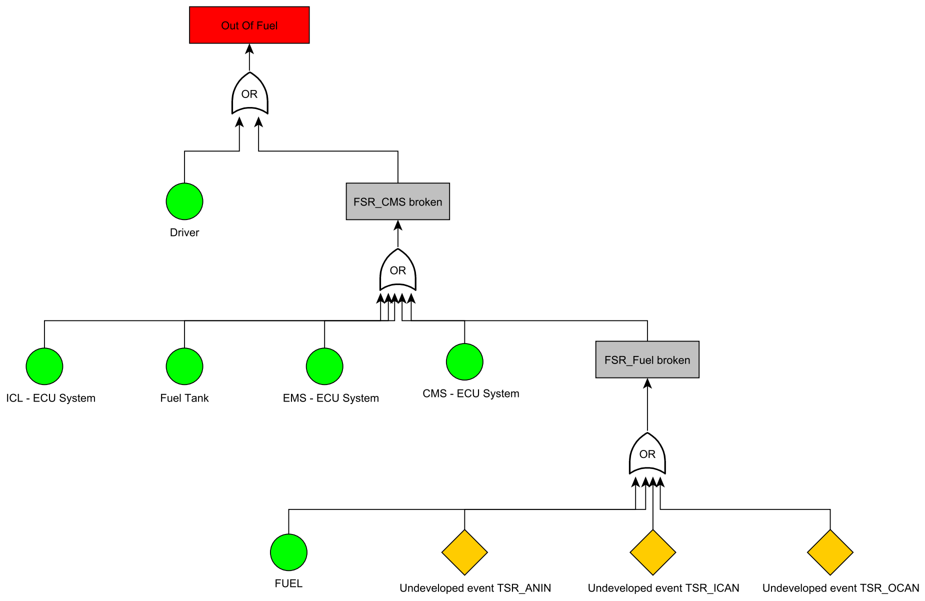

Figure 5.3: *The fault tree describing the third level of fuel level displays safety requirement*

Level 4. Safety requirements on BIOS SW components

BIOS-components manages the interaction between software and hardware component e.g. by sending signals corresponding to voltage values to the MIDD-components.

In accordance with earlier levels, the requirements TSR_{ADCC}, TSR_{RCAN}, SSR_{RCAN} and TSR_{TCAN} will be seen as OR-gates with their assumptions and elements as inputs and the requirements without assumptions, SSR_{DMAC}^{1}, SSR_{DMAC}^{2}, SSR_{BUFF}^{1} and SSR_{BUFF}^{2}, will have their allocated elements as basic events.

Next, logic gates with more than one of the same basic events will have the redundant basic events removed. The results of the fault tree created from four levels of safety requirements is seen in Figure 5.4.

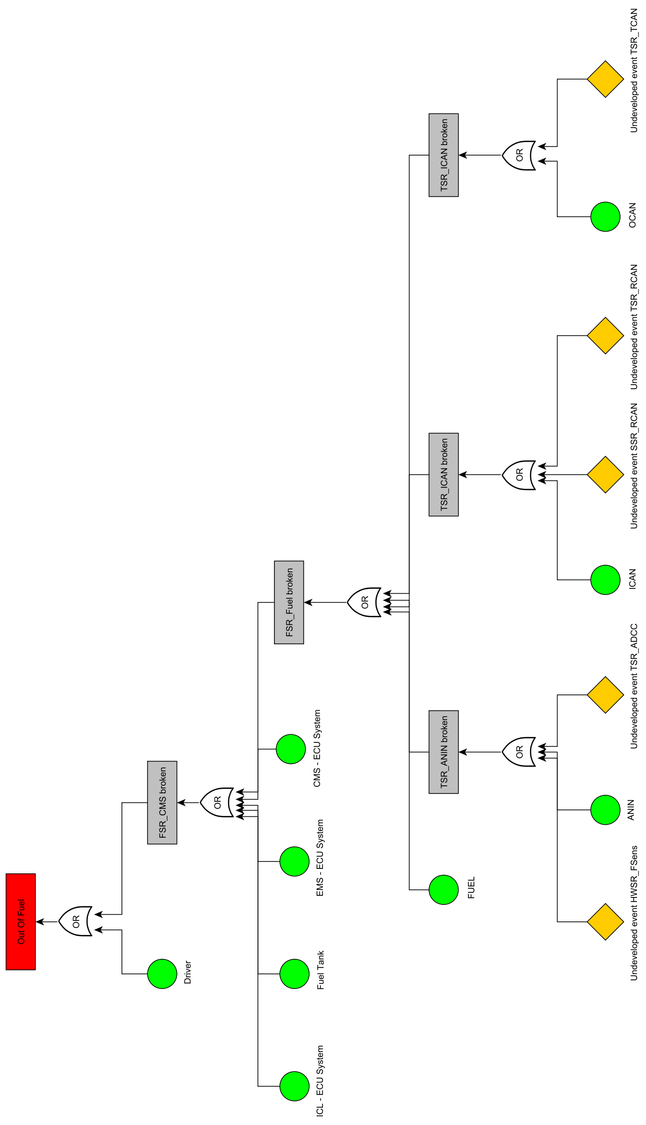

Figure 5.4: *The fault tree describing the fourth level of fuel level displays safety requirement*

Level 5. Safety requirements on COO ECU HW or HW components

The fifth level of safety requirements are requirements on the hardware components allocated to the item, e.g. the fuel sensor. All the safety requirements on the fifth level is hardware components that lack assumptions of other requirements. Therefore they will all be turned into basic events in the fault tree, which now is complete for the fuel level display. Figure 5.5 shows the complete fault tree of the fuel level display.

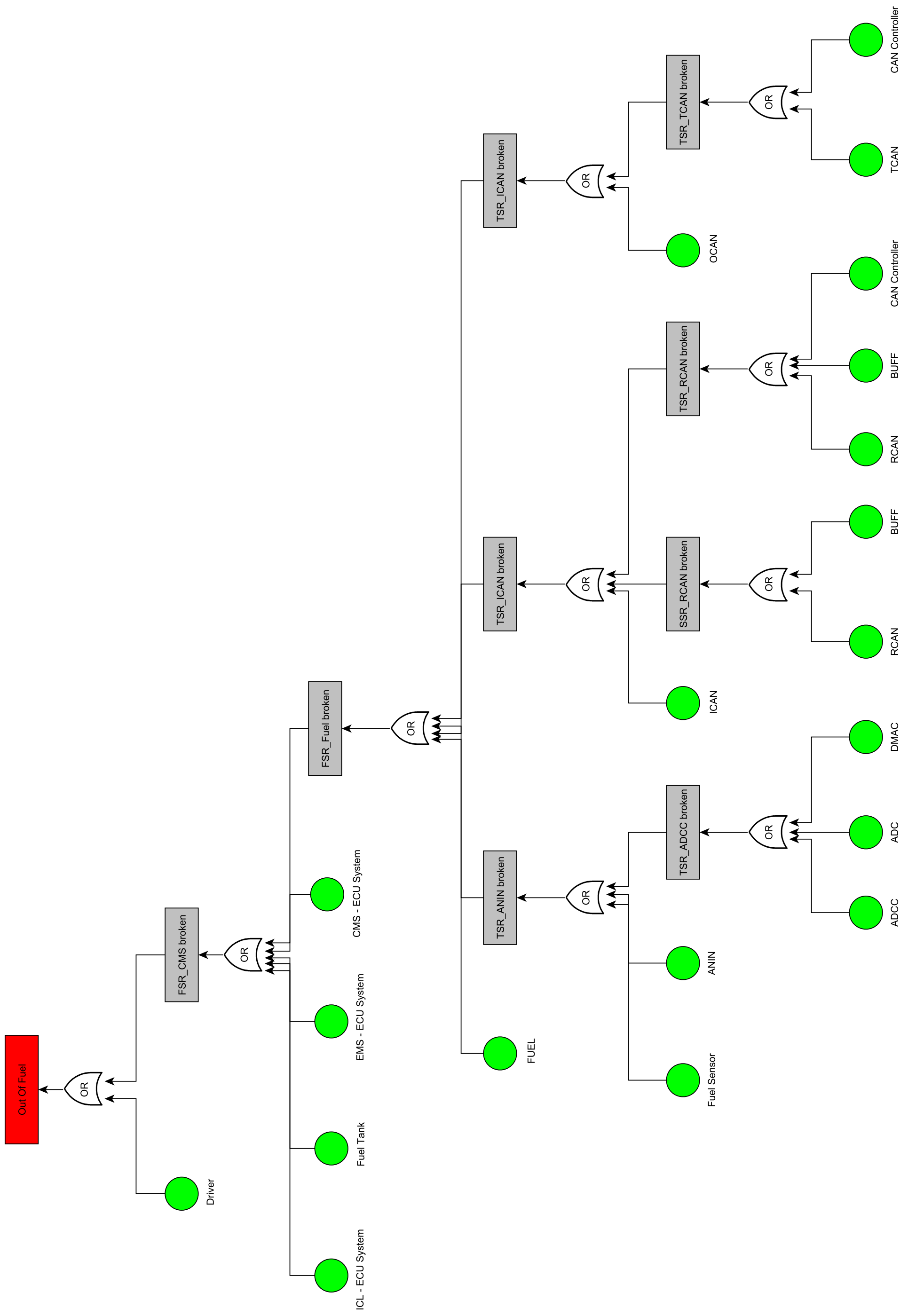

Figure 5.5: *The completed tree that describes the fuel level display*

5.2 Results evaluation: Dual circuit steering

Evaluation of the dual circuit steering is done using the same methodology as for the FLD. The hazard that is analyzed in this system is NoServoSteering, which means that the driver will have trouble to make the vehicle turn.

5.2.1 Dual circuit steering manual fault tree generation

The information about the requirements of dual circuit steering is not as extensive as the requirements of fuel level display, which causes the system description to be less accurate. The main purpose of using the requirement structure of dual circuit steering is to show how redundancy in the requirement structure can be described as well as how it will be translated into the fault tree. Therefore the failure mode propagation is not the focus in this case study. Figure 6.6 shows the generated fault tree of the dual circuit steering.

Level 1. SESAMM function level

This subsection considers the requirements that are responsible for the functional part of the dual circuit steering and is part of the requirement of the dual circuit steering. The requirements are taken from the requirement structure of dual circuit steering displayed in Table 3.6.

The failure mode propagation in this example system is very easy, the failure modes are no_Fault and signal_value_faulty. Table 5.2 displays the failure mode propagation in dual circuit steering and says that, unless anything else is mentioned, the failure mode of the child node is Faulty if any of the parents have the Faulty failure mode. This means that the hazard **NoServoSteering** is caused by the failure mode signal_value_faulty in the safety goal.

Table 5.2: *The failure mode propagation in DCS, with the parents in {parent}.*

Failure Mode	Parent Failure Mode Combinations
NF	{1}NF AND {2}NF AND ⋮ AND {n}NF
signal_value_faulty	{1}signal_value_faulty OR {2}signal_value_faulty OR ⋮ OR {n}signal_value_faulty

The safety goal states that the truck must be able to control during nominal driving. For that to work it is fulfilled by the requirement SubGoal1 as well as assuming that the requirement on the mechanical steering system MechSteeringReq1 is fulfilled. It will be developed as a gate with two inputs.

The MechSteeringReq1 is a requirement on the mechanical steering of the truck and since it is not part of the E/E systems of the truck there is no further development of the requirement, i.e., it has no assumptions and the element Mechanical Steering will be a basic event in the fault tree.

SubGoal1 fulfills the SG and has the assumptions from SubGoal2 and PrimaryCiruitReq1, where there exist redundancy between the two assumptions, which means that only one of the requirements has to be fulfilled for SubGoal1 to be fulfilled. This will create an AND-gate with two inputs. PrimaryCircuitReq1 has no further breakdown, i.e., no assumptions which means that the element Primary Hydraulic Flow Circuit will be a basic event in the fault tree.

Subgoal2 fulfills the requirement SubGoal1 and is in turn fulfilled by the SubGoal3. It also has the assumption DriverReq1, which is an requirement with no assumptions and the element Driver. The OR-gate that represents the SubGoal2 in the fault tree has two inputs, one from the requirement SubGoal3 and one from the basic event Driver.

The last sub goal, SubGoal3, fulfills SubGoal2. It has two assumptions on the SESAMM of the truck. The assumptions SESAMMReq1 and SESAMMReq2 are allocated to the

Chassis Management System (CMS) and the Secondary Hydraulic Circuit. SESAMMReq1 and SESAMMReq2 will be developed in further in dual circuit steering level 2. Figure 5.6 shows the fault tree resembling the first level of safety requirements on dual circuit steering.

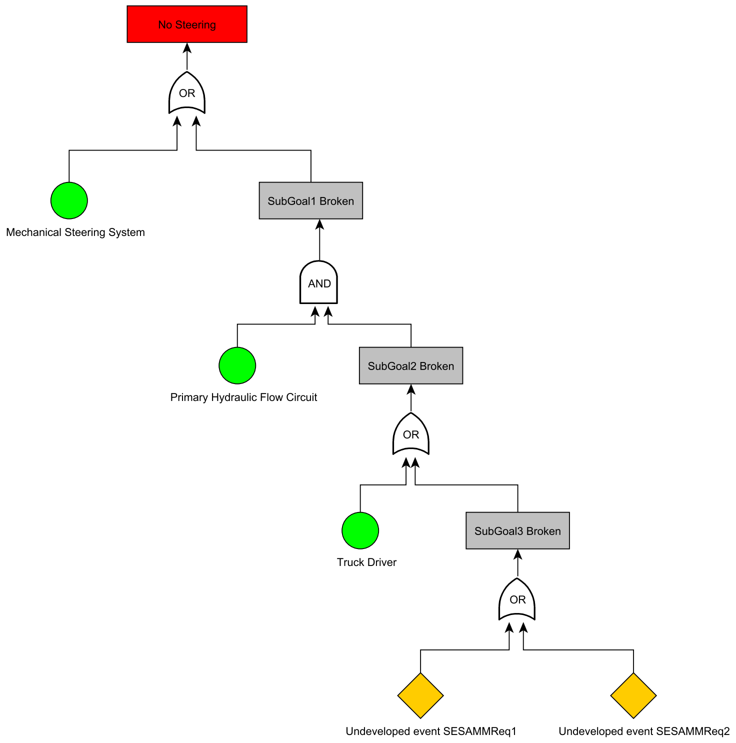
No Steering
OR
Mechanical Steering System
SubGoal1 Broken
AND
Primary Hydraulic Flow Circuit
SubGoal2 Broken
OR
Truck Driver
SubGoal3 Broken
OR
Undeveloped event SESAMMReq1
Undeveloped event SESAMMReq2

Figure 5.6: *The manually created fault tree describing the first level of safety requirements of the system dual circuit steering*

Level 2. AE421 level

Safety requirements on level 2 are requirements for the item, responsible for the back up circuit if the primary flow circuit does not work, and its environment. The safety requirements of level 2 are taken from Table 3.7. SESAMMReq1 has the allocated element CMS as well as 8 assumptions and SESAMMReq2 has the item Secondary Hydraulic Flow Circuit and 8 assumptions. The rest of the requirements on level 2 has only allocated elements and no assumptions, which means that their elements will be basic events in the fault tree. This means that there is only the Infrastructure requirements left that is not developed yet, since they are part of level 3 safety requirements. The results of the fault tree created from the first two levels of safety requirements is displayed in Figure 5.7.

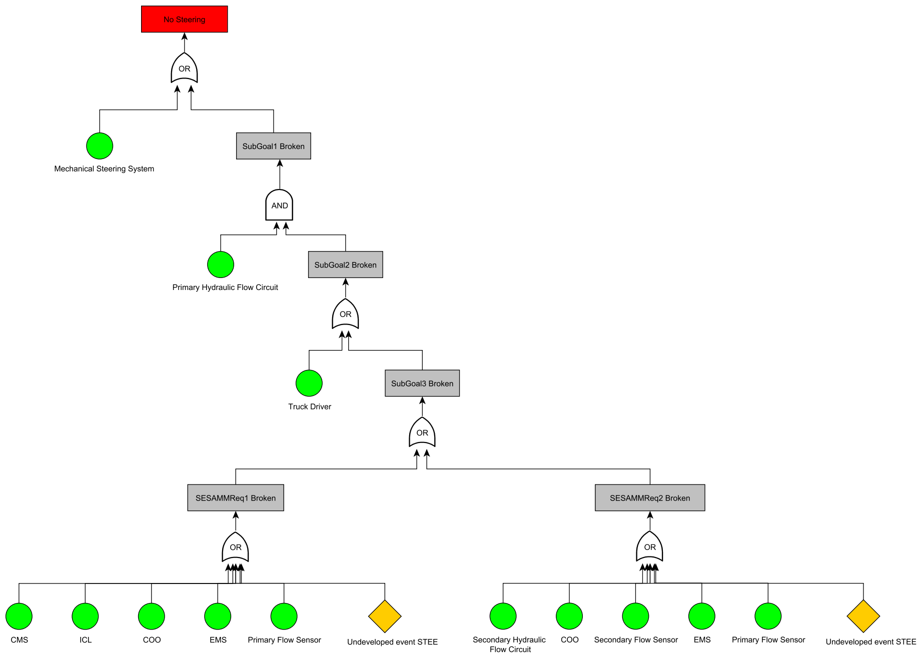

Figure 5.7: *The manually created fault tree describing two first level of safety requirements of dual circuit steering*

Level 3. Infrastructure software level

The infrastructure requirements that exists in level 3 of the safety requirements is taken from Table 3.8. None of the infrastructure requirements have any assumptions, which means that their elements will be basic events in the fault tree. The allocated element is STEE for all the requirements which means that many of the requirements will give redundant basic events. The redundant basic events will be removed from the tree since the same safety requirement on the same level in the fault tree does not give any additional information about the system, see Theorem 4.3. Figure 5.8 shows the final manually created fault tree of the dual circuit steering.

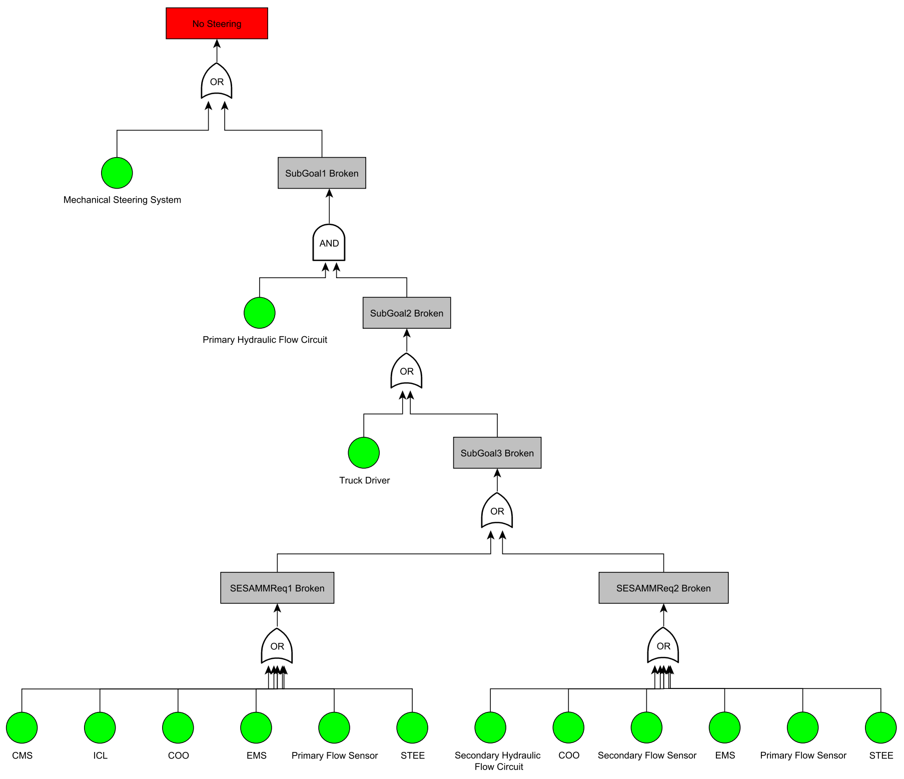

Figure 5.8: *The manually created fault tree describing all three levels of of safety requirements describing dual circuit steering*

6 Results

This chapter is dedicated to show the case study of the thesis. The results concern the generation of fault trees and failure propagation networks, but also results regarding the extension of the used database and how the definition of failure mode propagation is done.

6.1 Bayesian network evaluation

The generated Bayesian networks of the system fuel level display is presented in Figure 6.1. A close up on the hazards and safety goals can be seen in Figure 6.2. It is shown that the hazard OutOfFuel is active (the OutOfFuel node has 100% probability to be active) and which requirements and elements are affecting it. The network also has the ability to introduce a failure mode in any node and evaluate which nodes are affected by it.

Figure 6.3 shows the Bayesian network that handles the failure propagation for DCS. As opposed to FLD, DCS has redundancy in the system and this is presented by using an OR node in the network that connects the requirement on the primary circuit together with the requirement SubGoal2. The redundancy can be seen in Figure 6.4 by setting the hazard NoServoSteering to active as well as setting evidence of a fault in a hydraulic flow sensor. The Bayesian network then says that it has to be a fault in either the mechanical steering system or in the primary hydraulic flow circuit.

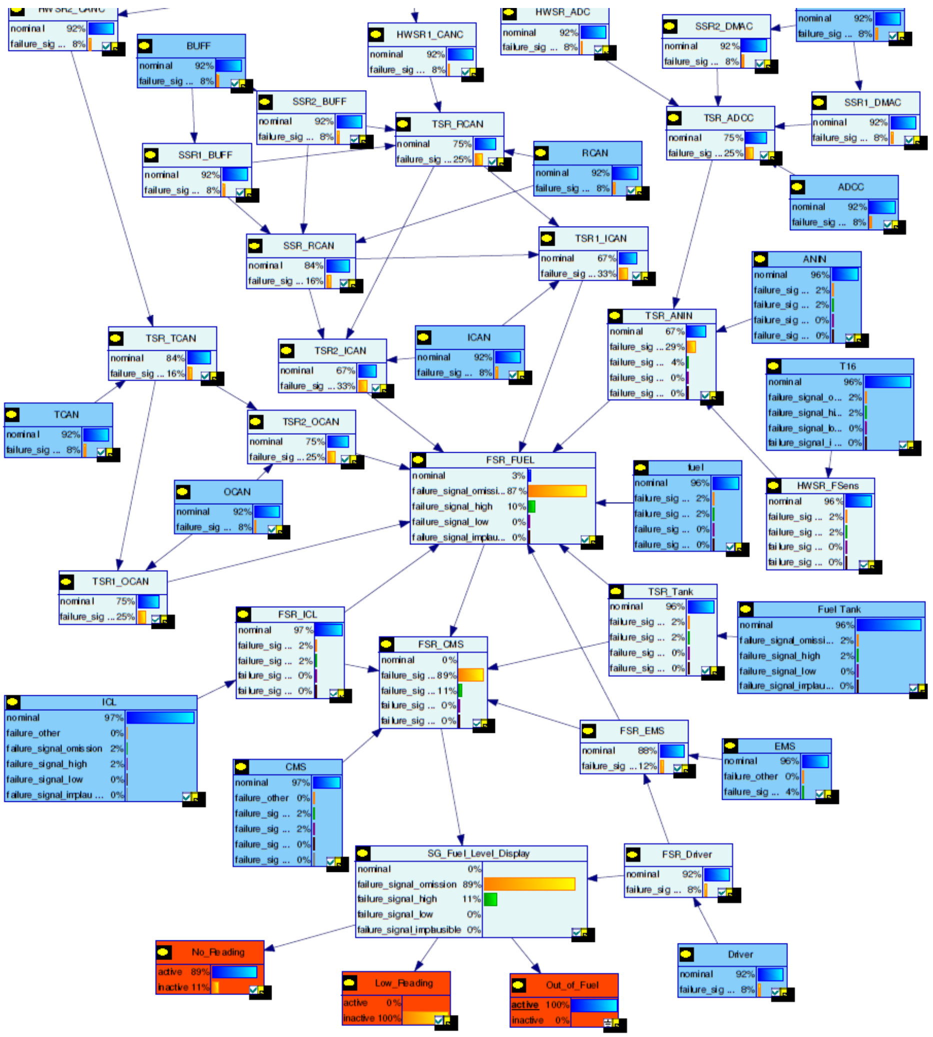

Figure 6.1: *Bayesian network describing the failure mode propagation in fuel level display. The red squares resemble the system hazards. The bars show the conditional probabilities in the network.*

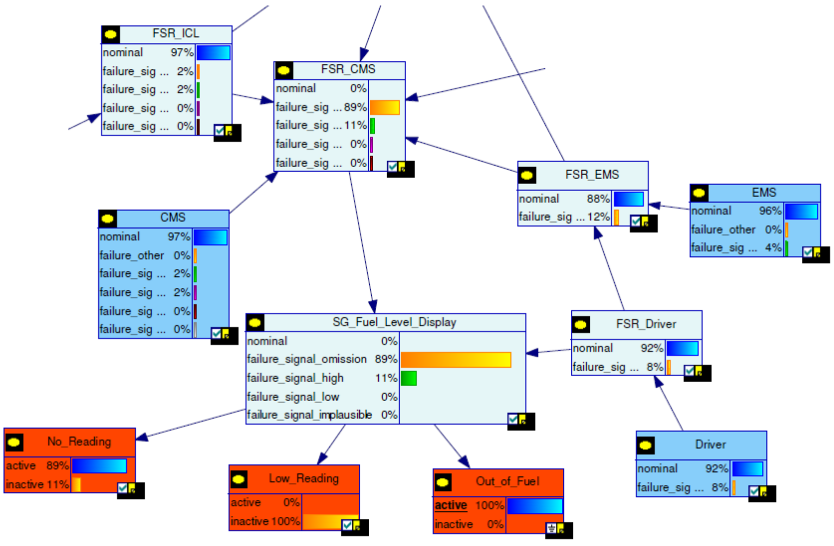

Figure 6.2: *A close up on the hazards and safety goal in the fuel level display. It is possible to observe that the hazard OutOfFuel and it's effect on adjacent nodes.*

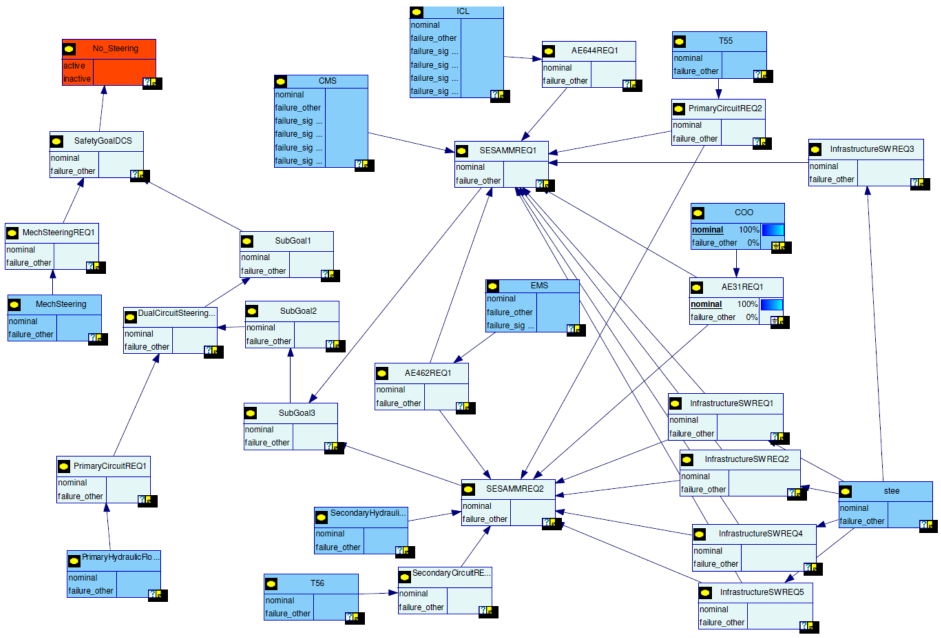

Figure 6.3: *Bayesian network describing the failure mode propagation in dual circuit steering. No probabilities are set in the network.*

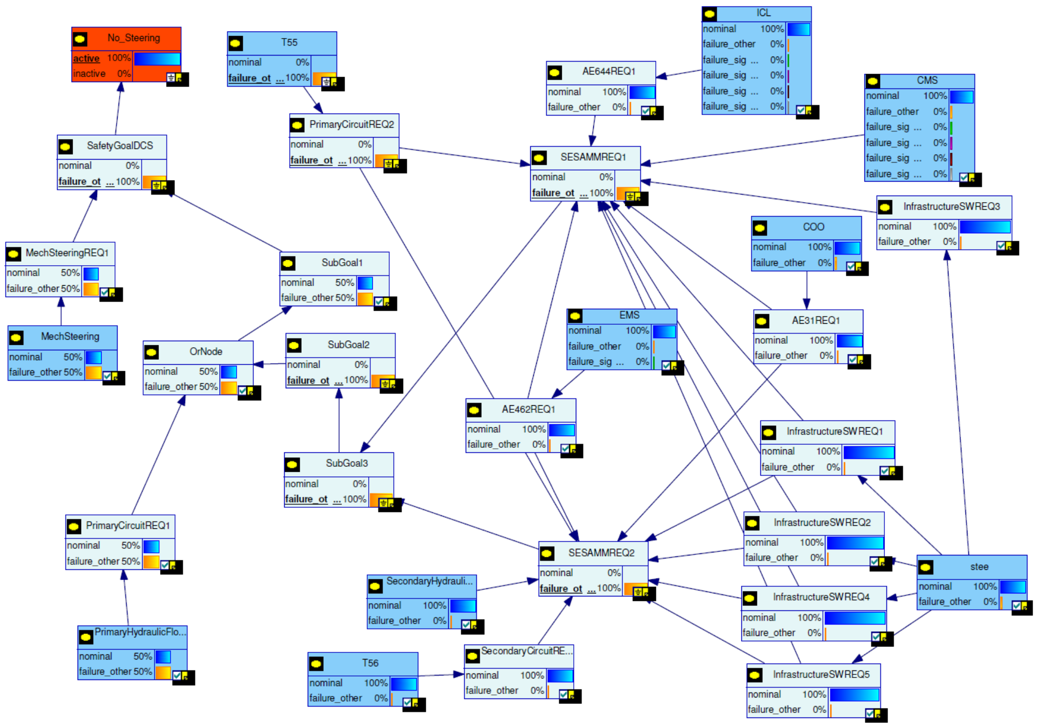

Figure 6.4: Bayeisan network illustrating the effects of setting the probability of the hazard to 1. It is also possible to see the effects of redundancy in the network.

6.2 Fault tree generation evaluation

The fault trees for the systems is evaluated by comparing the automatically and manually generated fault trees in the case study. This is done to validate the fault tree generation method.

6.2.1 Results evaluation: Fault tree comparison

The manually created fault tree in Figure 5.5 tree can now be compared against the automatically generated tree in Figure 6.5. The two fault trees are identical on every level, which is explained by the fact that the same information is used when generating the trees.

The difference between the trees when creating the trees manually and automatically generating them is that the requirement description is not used when automatically generating the trees. The requirement descriptions have information how the guarantee behaves depending on the assumptions and the allocated element, e.g., by saying that if an error value is received then a warning will be shown to the driver. This will result in a degraded state, i.e., a faulty but safe state. The error handling works, in a safety perspective, similar to redundancy in a system.

Since the error handling only is described in the requirement description, the ability to retrieve that information is somewhat difficult, since that data is not retrieved from the database when building the fault trees. It is possible to model error handling in the failure propagation network by adding an error failure mode. It is also possible to add extra elements responsible for the error handling to the propagation network. Then OR nodes, similar to Definition 2.20, can be created connecting error handling elements and other nodes to simulate redundancy.

Error handling are often based on the system finding out that something is wrong, e.g., by discovering that some values are implausible/unlikely or does not match another value. Even if it is possible to model error handling in the Bayesian network it is not always possible to describe it in the fault tree. According to [8, p. 177]: "Fault trees have long been used for reliability modeling because of their concise representation of system structure, but they cannot adequately model the dynamic system behavior in response to a fault or error." Because of this, error handling is not dealt with in this thesis.

As well as automatically generating fault trees it is also possible to generate minimal cut sets of the fault tree and using it to generate a minimal tree of the system describing the minimal sets of basic events that can cause a hazard. The minimal tree for OutOfFuel can be seen in Figure 6.8. Some fault tree analysis can be done on the generated fault trees to generate a minimal cut set, or a minimal tree describing the minimal failure causes of the hazard.

Figure 6.6 shows the generated version of dual circuit steering which can be compared to the manually created fault tree in Figure 5.8. The fault trees has identical basic events building up the tree, which is to be expected since they are based on the same requirements and elements. Also the structure of the fault tree is the same with the same basic events on every level. This is also expected because a similar procedure is done when creating the trees manually compared to generating them.

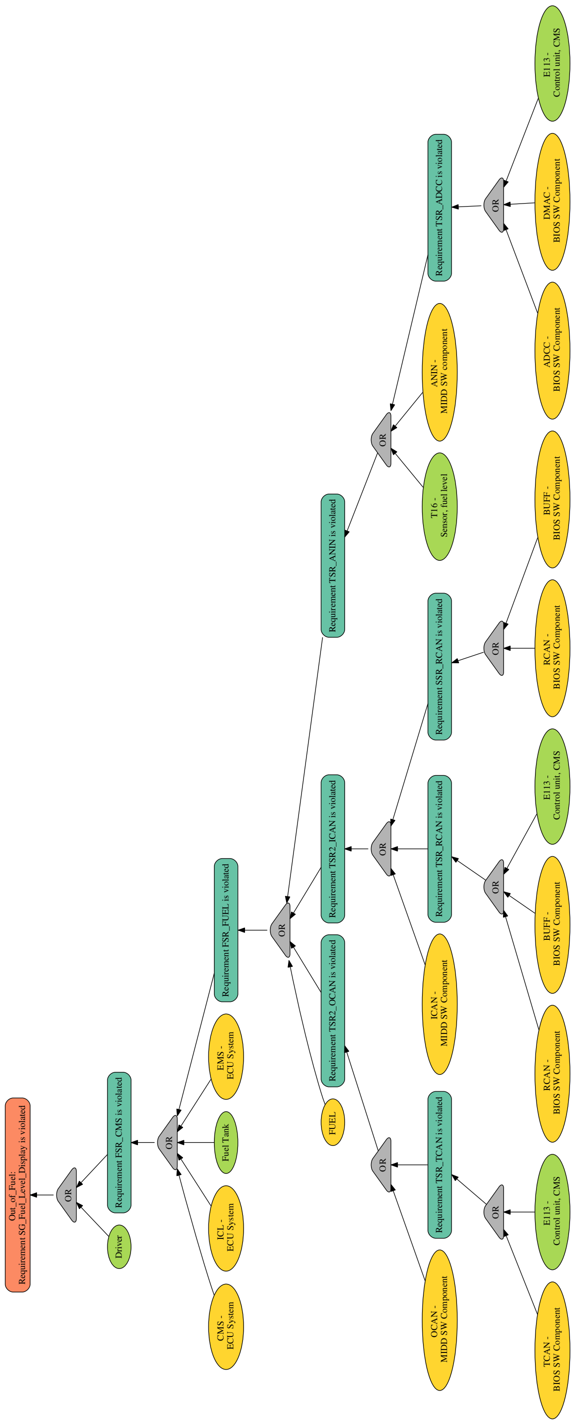

Figure 6.5: *A generated version of the fault tree describing the system fuel level display. The top event is the hazard OutOfFuel.*

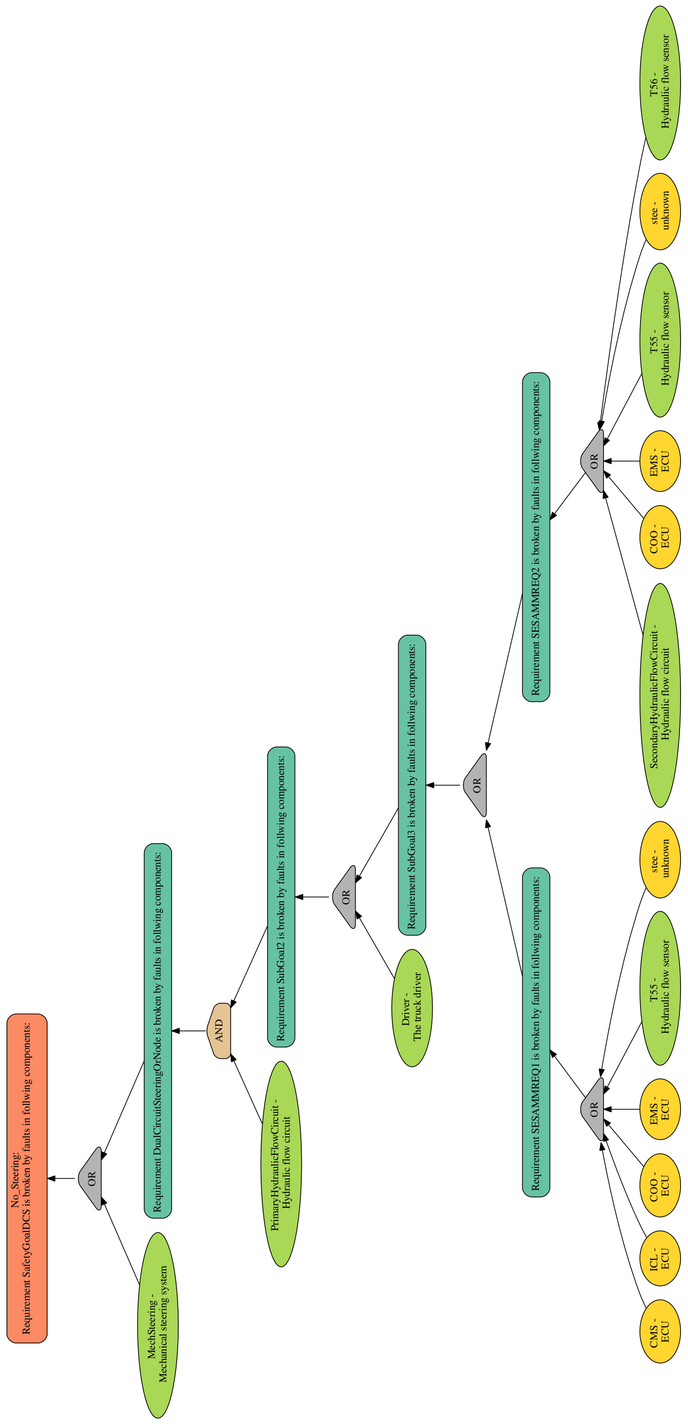
No_Steering:
Requirement SafetyGoalDCS is broken by faults in follwing components:
OR
Requirement DualCircuitSteeringOrNode is broken by faults in follwing components:
MechSteering -
Mechanical steering system
AND
Requirement SubGoal2 is broken by faults in follwing components:
PrimaryHydraulicFlowCircuit -
Hydraulic flow circuit
OR
Requirement SubGoal3 is broken by faults in follwing components:
Driver -
The truck driver
OR
Requirement SESAMMREQ2 is broken by faults in follwing components:
Requirement SESAMMREQ1 is broken by faults in follwing components:
OR
OR
T56 -
Hydraulic flow sensor
stee -
unknown
T55 -
Hydraulic flow sensor
EMS -
ECU
COO -
ECU
SecondaryHydraulicFlowCircuit -
Hydraulic flow circuit
stee -
unknown
T55 -
Hydraulic flow sensor
EMS -
ECU
COO -
ECU
ICL -
ECU
CMS -
ECU

Figure 6.6: *A fault tree of the system dual circuit steering. The top event is the hazard NoServoSteering.*

6.2.2 General discussion

From the results of the fuel level display and dual circuit steering when comparing the automatic generated and manually generated fault trees it seems like the fault tree generation is working as intended when starting from a requirement structure.

The last thing to examine is if the fault trees that are generated from the requirement structure are the same as a manually generated fault trees that are created from knowledge of the system, i.e. the way fault trees usually are created. If the results are the same it is possible to say if the data from the requirement structures, together with failure mode propagation, are sufficient enough data to generate fault trees.

The fault tree in Figure 6.7 is created with help of Scania employee Magnus Sellldén. The fault tree is similar to the automatically generated fault tree in Figure 6.5. Both systems have no redundancy and focuses on the ECU-system CMS. The fuel sensor, the floater in the tank, the fuel filter as well as the communication software are similar in both fault trees.

The major differences between the expert created tree and the tree created from requirements is that the fault tree generated from the requirement structure is further broken down into BIOS and hardware elements that the manual fault tree is not. Another distinction is that the generated fault tree does not have any break down on the ICL and EMS ECU-system which the manual tree has. This means that the differences between the trees are mostly due to which parts of the system that are broken down, which will result in the trees to have finer resolution on some parts compared to the other tree.

The reason that different part of the systems are broken down depends on which parts that are important to the creator. The experts main focus is on the functional levels of the system, i.e., level 1-3.. Also some of the functional requirements from other ECU systems where broken down further. The creator of the requirement structure also focuses on the lower levels, 4 and 5, which explains the small difference in the generated fault trees.

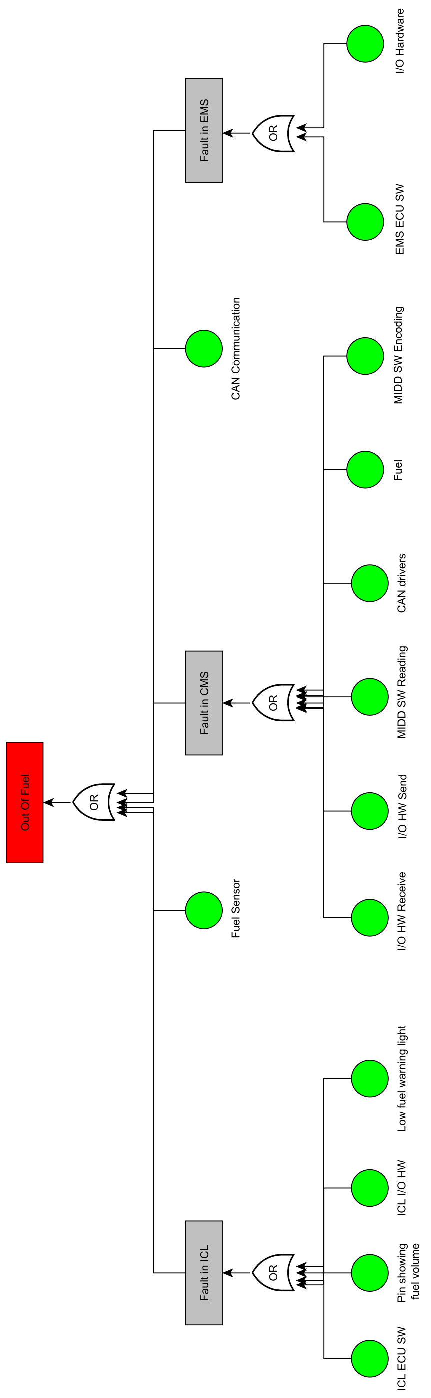

Figure 6.7: *A fault tree describing fuel level display created with help of Scania employee Magnus Selldén.*

Some phenomenons that can be hard to get both from the requirement structure and model in the failure propagation network are so called cascade failures. A cascade failure means that more than, e.g., sending a faulty signal the failure creates faults in other elements [20], i.e., a broken resistor in a circuit can make another resistor break because of increased current passing through. This is a phenomenon that is not described in any of the requirement structures (FLD/DCS). This phenomenon is also hard to model in the failure propagation network since elements that are faulty does not affect other elements, i.e., cascade failure can not be modeled in the current way the propagation network is constructed. The minimal tree for dual circuit steering can be seen in Figure 6.9.

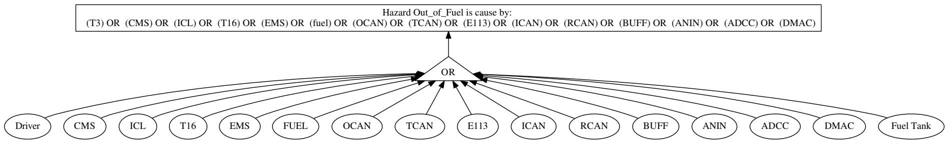
Hazard Out_of_Fuel is cause by:
(T3) OR (CMS) OR (ICL) OR (T16) OR (EMS) OR (fuel) OR (OCAN) OR (TCAN) OR (E113) OR (ICAN) OR (RCAN) OR (BUFF) OR (ANIN) OR (ADCC) OR (DMAC)
OR
Driver
CMS
ICL
T16
EMS
FUEL
OCAN
TCAN
E113
ICAN
RCAN
BUFF
ANIN
ADCC
DMAC
Fuel Tank

Figure 6.8: *The minimal tree generated from the minimal cut sets of the hazard OutOfFuel.*

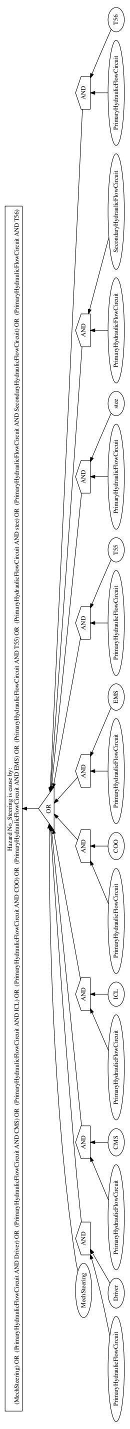
Hazard No_Steering is cause by:
(MechSteering) OR (PrimaryHydraulicFlowCircuit AND Driver) OR (PrimaryHydraulicFlowCircuit AND CMS) OR (PrimaryHydraulicFlowCircuit AND ICL) OR (PrimaryHydraulicFlowCircuit AND COO) OR (PrimaryHydraulicFlowCircuit AND EMS) OR (PrimaryHydraulicFlowCircuit AND T55) OR (PrimaryHydraulicFlowCircuit AND stee) OR (PrimaryHydraulicFlowCircuit AND SecondaryHydraulicFlowCircuit) OR (PrimaryHydraulicFlowCircuit AND T56)
OR
AND
MechSteering
Driver
PrimaryHydraulicFlowCircuit
CMS
ICL
COO
EMS
T55
stee
SecondaryHydraulicFlowCircuit
T56

Figure 6.9: *The minimal tree generated from the minimal cut sets of the hazard NoServoSteering.*

7 Conclusion

This chapter concludes the thesis and discusses the results as well as future works.

7.1 Conclusion

Since the generation of fault trees is automatic, it is a straightforward and time saving method if the failure propagation and safety requirements already exist. If one or both does not exist, this method for generating fault trees may not be either straightforward nor time saving. But since the ISO26262-standard requires safety requirements, and information about failure modes is needed for FMEA, it is likely that both will exist at Scania for most systems in the future.

Because all the requirement information cannot be parsed and some faults are not described by the requirements, the generated fault trees does not always describe the system entirely. Although, if branches has to be added to complete the fault tree it can also be an indication that the safety requirements is not fully comprehensive. Since the quality of the fault trees are very dependant on the requirements, i.e., if they are inadequate or wrong this will be translated directly to the fault trees. Using this, it is possible to validate that safety requirements are correctly written. This can be done in system where fault trees already exists, and the verification can be done by comparing existing trees with fault trees generated from requirements. If the trees differ it can be an indication that the requirements for the system are inadequate.

Since minimal cut sets are automatically generated it is possible to calculate a failure probability in a system, if the fault probability in all included elements can be estimated. This calculation gives a fast and rough estimation of how safe the system is. Which can be used to evaluate if, e.g., measures has to be taken to lower the failure rate.

Bayesian networks are used to describe the failure propagation in systems to be able to create the fault trees. There exist programs for failure propagation, e.g, HiP-HOPS, but since it does not work in this case, Bayesian networks were used instead. A limitation with Bayesian networks is that the required computational power grows fast with the number of nodes and states in the network, which can make large failure propagation networks hard to compute. But since the computation is done offline and no time limitation this is no real limitation in most cases. If the computational time is a problem there are methods for decreasing the complexity in CPTs which could be implemented to reduce the calculation intensity if needed. Another benefit with the Bayesian networks are that they also can be used to generate FMEA-tables, i.e. the other master thesis in this project. The Bayesian networks can also be used for diagnosis for the system. This can be done if probabilities for different faults as well as the probability of faults causing specific failure modes are introduced into the network.

When it comes to validation of the fault trees some difficulties occurred. Since it did not exist any fault trees describing the example system to compare against, the verification of the fault tree generation had to be done in two steps. First, the method of generating fault trees from the requirement structure had to be validated, i.e., to see if the method works as intended given the requirements. Second, the finished fault trees had to be verified against the real system, i.e., investigating if the information taken from the requirement structures is enough to generate accurate fault trees. The verification was done by first generating fault trees from the requirement structures manually and then compare them to the automatically generated fault trees. The validation certainty could have been increased if the manual fault trees had been created by an expert with knowledge of safety requirements as well as the actual system. The second verification was done with the help of a Scania expert where we together created a fault tree for the system fuel level display. Since fault trees are not a safety analysis method used at Scania, the employees are not used to the method, this was not an optimal scenario for the validation.

When starting the master thesis, data describing the failure mode propagation for the two systems did not exist. Therefore the failure propagation had to be created from information found in FMEA-tables of the two systems. Since that information were not enough to describe the entire failure propagation, this can be seen as an additional possible source of errors in the validation.

The trees that are automatically generated in this thesis only consists of the static AND and OR gates. This means that no dynamic behaviour is described, e.g., behaviour which considers in which order events happen. Also not using voting gates, i.e., gates that returns true if x out of y inputs are true makes the fault trees larger and a bit more complicated than it could be. The extension of the latter is pretty straight forward to implement. Including dynamic gates would on the other hand change the generation process.

Some faults, e.g., cascade failures and common cause failures (CCF) is not included in the fault trees since those types of failure is not described by the requirement structures. This is a weakness of this method, but the CCFs can be added to the fault trees after the automatic generation.

In this thesis it is shown that it is possible to generate a success tree using only a requirement structure that is structured using contract theory as base. To create a fault tree, additional information how the different faults translate into different hazards, for this failure propagation is added. The failure propagation is needed because otherwise it is not possible to say which hazard/hazards that can be caused, only that the system is not safe.

7.2 Future work

As a future work it can be investigated how the Bayesian network generated from requirement structures can be used as a diagnostic tool, e.g., by introducing probabilities of different faults given broken requirements. It can also be investigated if the network can be used in a method of verifying the safety requirements. Finally a method for generating the failure mode propagation, e.g from system models, could simplify the process even further.

If the safety requirements were written in a more formal way, the information in the requirements could be used by the program when generating the Bayesian network as well as the fault trees. A future work could be to use that information to improve the generation of the trees, e.g., when considering error handling or other properties described in the requirement. Extending that would probably require some extension to the fault tree generation to include dynamic gates, e.g., priority-AND which means that the gate returns true if and only if inputs happen in a predetermined order [7]. This is needed since the error handling has to be the system that crashes first in order for the failure to go unnoticed. The extension of dynamic gates and dynamic fault tree analysis could be another thesis work based on this thesis.

Another continuation could be to investigate if it is possible to generate safety requirements, or at least templates, from a failure propagation network. Where the networks could be used to show which elements that needs requirement definitions as well as knowing how elements affect each other.

Bibliography

[1] ISO 26262-1:2011. Road vehicles — functional safety — part 1: Vocabulary, 2011. Cited on pages 3, 4, 9, and 10.

[2] ISO 26262-4:2011. Road vehicles — functional safety — part 4: Product development at the system level, 2011. Cited on pages 1 and 4.

[3] ISO 26262-9:2011. Road vehicles — functional safety — part 9: Automotive safety integrity level (asil)-oriented and safety-oriented analyses, 2011. Cited on page 5.

[4] A. Bobbio, L. Portinale, M. Minichino, and E. Ciancamerla. Improving the analysis of dependable systems by mapping fault treesinto Bayesian networks. *Reliability Engineering & System Safety*, 71:249–260, 2001. Cited on pages 6 and 19.

[5] Andrea Bobbio, Luigi Portinale, Michele Minichino, and Ester Ciancamerla. Comparing fault trees and bayesian networks for dependability analysis. In Massimo Felici and Karama Kanoun, editors, *Computer Safety, Reliability and Security*, volume 1698 of *Lecture Notes in Computer Science*, pages 310–322. Springer Berlin Heidelberg, 1999. Cited on page 6.

[6] Hichem Boudali, Pepijn Crouzen, and Mariëlle I.A. Stoelinga. Dynamic fault tree analysis using input/output interactive markov chains. In *In The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, 25-28 June 2007, Edinburgh,UK, Proceedings*, pages 708–717, 2007. Cited on page 5.

[7] Marc Bouissou. A Generalization of Dynamic Fault Trees through Boolean logic Driven Markov Processes (BDMP)®. In *Proceedings of ESREL 2007*, 2007. Cited on page 81.

[8] B J Dugan. Fault trees and imperfect coverage. Technical report, Durham, NC, USA, 1987. Cited on pages 23 and 71.

[9] ESPRESSO. Metamodel Neo4J Database, 2015. Cited on page 34.

[10] F. Hu. *Cyber-Physical Systems: Integrated Computing and Engineering Design*. Taylor & Francis, 2013. Cited on page 10.

[11] Finn V. Jensen and Thomas D. Nielsen. *Bayesian Networks and Decision Graphs.* Springer Publishing Company, Incorporated, 2nd edition, 2007. Cited on page 15.

[12] Jan Karlsson. FMEA CMS1 (PD523329), 2014. Cited on pages 22, 23, and 53.

[13] Decision Systems Laboratory. GeNIe & SMILE. `https://dslpitt.org/genie/`, 2015. [Online; accessed 22-April-2015]. Cited on pages 17, 42, and 44.

[14] P. Liggesmeyer and M. Rothfelder. Improving system reliability with automatic fault tree generation. In *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pages 90–99, June 1998. Cited on page 6.

[15] Xiaofeng Liu and Siqi An. Failure propagation analysis of aircraft engine systems based on complex network. *Procedia Engineering*, 80(0):506 – 521, 2014. 3rd International Symposium on Aircraft Airworthiness (ISAA 2013). Cited on page 6.

[16] Inc. Neo Technology. Neo4J product information, 2014. Cited on page 34.

[17] M. Nyberg. Failure propagation modeling for safety analysis using causal bayesian networks. In *Control and Fault-Tolerant Systems (SysTol), 2013 Conference on*, pages 91–97, Oct 2013. Cited on pages 5 and 6.

[18] Yiannis Papadopoulos, Martin Walker, David Parker, Erich Rüde, Rainer Hamann, Andreas Uhlig, Uwe Grätz, and Rune Lien. Engineering failure analysis and design optimisation with hip-hops. *Engineering Failure Analysis*, 18(2):590 – 608, 2011. The Fourth International Conference on Engineering Failure Analysis Part 1. Cited on page 42.

[19] United States. Department of Defense. *Mil-Std-1629a: 1980: Procedures for Performing a Failure Mode, Effects and Criticality Analysis.* Department of Defense, 1980. Cited on page 3.

[20] W. Vesely, J. Dugan, J. Fragola, Minarick, and J. Railsback. Fault Tree Handbook with Aerospace Applications. Handbook, National Aeronautics and Space Administration, Washington, DC, 2002. Cited on pages 3, 19, 20, 39, and 76.

[21] Malcolm Wallace. Modular architectural representation and analysis of fault propagation and transformation. In *Proc. FESCA 2005, ENTCS 141(3), Elsevier*, pages 53–71, 2005. Cited on page 6.

[22] J. Westman and M. Nyberg. Extending contract theory with safety integrity levels. In *High Assurance Systems Engineering (HASE), 2015 IEEE 16th International Symposium on*, pages 85–92, Jan 2015. Cited on pages 10, 12, 15, and 24.

[23] Jonas Westman. Breakdown of Safety Goal to Software Requirements Power Point Presentation Demo Day January 2015, 2015. Cited on page 7.

[24] Jonas Westman and Mattias Nyberg. A reference example on the specification of safety requirements using iso 26262. In *Proceedings of Workshop DECS*

(ERCIM/EWICS Workshop on Dependable Embedded and Cyber-physical Systems) of the 32nd International Conference on Computer Safety, Reliability and Security - SAFECOMP 2013 - Workshop DECS (ERCIM/EWICS Workshop on Dependable Embedded and Cyber-physical Systems) of the 32nd International Conference on Computer Safety, Reliability and Security, France (2013) :, 2013. NQC 20140128. Cited on page 6.

[25] Jonas Westman and Mattias Nyberg. Specifying and structuring requirements on cyber-physical systems using contracts. Technical report, KTH, Mechatronics, 2014. QS 2015. Cited on pages 6, 7, 10, 12, 13, 14, 21, 22, 24, 25, and 27.

[26] Jonas Westman, Mattias Nyberg, and Martin Törngren. Structuring safety requirements in iso 26262 using contract theory. In *Computer Safety, Reliability, and Security : 32nd International Conference, SAFECOMP 2013, Toulouse, France, September 24-27, 2013. Proceedings*, volume 8153 of *Lecture Notes in Computer Science*, pages 166–177, 2013. QC 20140128. Cited on page 10.