

CHECKING COMPLIANCE WITH ISO 26262 USING CONCEPTUAL MODELING AS A  
TOOL

CHECKING COMPLIANCE WITH ISO 26262

USING CONCEPTUAL MODELING

AS A TOOL

BY

SYEDA ZAHRA ALI NAQVI

McMaster University

© Copyright by Syeda Zahra Ali Naqvi, 2018

MASTER OF Department of Computing and Software  
APPLIED SCIENCE McMaster University  
1280 Main Street West  
Hamilton ON L8S 4K1

TITLE: Checking Compliance with ISO 26262  
using Conceptual Modeling  
as a Tool

AUTHOR: Syeda Zahra Ali Naqvi

SUPERVISOR: Dr. Thomas S.E. Maibaum

NUMBER OF PAGES: i – v, 1 – 103

YEAR: 2018

# Abstract

With the advancement in technology in the automotive sector, there is an increase in the need of safety assurance of road vehicles. ISO 26262 is a safety critical standard intended to be applied to safety-related E/E systems in passenger cars. The standard also addresses the possible hazards caused by the failure of these systems and uses a risk-based approach to ensure functional safety of the systems. It is important, during the design and development phases of a product, to ensure that the product is in compliance with the standard. Therefore, we present an idea of using conceptual modeling to help verify the E/E systems with ISO 26262. The standard categorizes the requirements and recommendations into what is known in the standard as a Work Product. Building conceptual models of these work products not only gives us a visual representation of the contents of a standard but also helps standardize the process of compliance checking of a product against the standard. The process of using conceptual modeling for checking compliance of a product with the standard is explained in this thesis. The technique of using conceptual modeling not only proved to be efficient for product verification against the standard but was also helpful in exposing important structural aspects of the standard.

# Acknowledgments

I would like to express my deepest gratitude to my supervisor Doctor Tom Maibaum for his mentorship and guidance throughout the project. I am equally grateful to Valentin Cassano for his unwavering support and supervision and for always providing valuable feedback for improvements in my work.

Special thanks to my family for providing me with unconditional love and support encouraging me to find my strengths, especially to my parents who have been my source of strength and a driving force throughout my years of study.

I would like to thank all those who in one way or the other, encouraged me to apply for this master's program and contributed towards my achievement of this milestone.

Last but not the least I would like to thank McMaster University for providing me with an incredible opportunity to pursue further education in the field of Software Engineering.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background and Motivation . . . . .	2
1.2	Objective . . . . .	3
1.3	Related Work . . . . .	4
1.4	Organization of the thesis . . . . .	6
<b>2</b>	<b>Basic Concepts of ISO 26262</b>	<b>7</b>
2.1	About the Standard . . . . .	7
2.2	Work Product . . . . .	8
2.3	Work Flow . . . . .	8
2.4	Conceptual Modeling . . . . .	11
2.5	Object Diagram . . . . .	14
2.6	Constraint Language . . . . .	15
2.7	Selection of Tools . . . . .	16
<b>3</b>	<b>Building the Conceptual Models</b>	<b>21</b>
3.1	Approach . . . . .	21
3.2	Selection of Work Products . . . . .	22
3.3	Creation of Class Diagrams . . . . .	23
3.4	Item Definition . . . . .	24
3.5	Hazard Analysis and Risk Assessment . . . . .	30
3.6	Safety Goals . . . . .	40
3.7	FLEDS - An Example . . . . .	45

3.8	Composition . . . . .	46
3.9	Allocation Elements . . . . .	47
3.10	FLEDS Requirements . . . . .	48
<b>4</b>	<b>Compliance with ISO 26262</b>	<b>53</b>
4.1	Approach . . . . .	53
4.2	Example Scenario . . . . .	54
4.3	Item Definition . . . . .	55
4.4	Hazard Analysis and Risk Assessment . . . . .	60
4.5	Safety Goals . . . . .	62
4.6	Compliance . . . . .	63
<b>5</b>	<b>Discussion</b>	<b>67</b>
5.1	Challenges and Limitations . . . . .	67
5.2	Conclusion . . . . .	69
5.3	Future Work . . . . .	71
<b>A</b>	<b>Appendix</b>	<b>72</b>
	<b>List of Acronyms</b>	<b>99</b>
	<b>References</b>	<b>102</b>

# Chapter 1

## Introduction

The advancement in technology and increasing demand of technological innovation in the automotive industry has led to the development of new and improved electrical and electronic systems, causing them to become more complex day by day. This has also resulted in an increased need to ensure safety of the automotive vehicles produced.

The International Organization for Standardization (ISO) is an international body composed of representatives from various countries that put together standards to define requirements, specifications, rules and guidelines that help in the production of products and services that are reliable and widely acceptable. One such standard formulated by the ISO is the ISO 26262 (ISO 26262, 2011) for the functional safety of the E/E systems in road vehicles.

The ISO 26262 is an adaptation of the IEC 61508, approved by the International Electrotechnical Commission (IEC) (Commission et al., 2000). IEC 61508 is a general safety critical systems standard, for functional safety of electrical/electronic/pro-



programmable electronic safety-related systems. ISO 26262 however focuses on providing guidelines and processes for the safety critical electrical, electronic and software systems in automobiles. It applies to safety-related systems that include one or more electrical and/or electronic (E/E) systems that are installed in series production passenger cars with a maximum vehicle mass up to 3,500 kg. The standard does not apply to the systems or components released in the market or under production prior to the release date of ISO 26262. ISO 26262 also addresses the hazards that could be caused by the malfunctioning of the safety related E/E systems or by the interaction of these systems to avoid or mitigate related risks. It does not however include the possible hazards that could be caused by accidental failures unless caused by the malfunctioning of these components.

## 1.1 Background and Motivation

For a safety critical system to be declared safe it has to undergo the process of certification. For a system to be certified safe, it has to follow some safety standard and a certification body has to declare that the system meets the requirements set forth by the standard and is safe to operate in a certain environment. A safety standard sets rules and regulations for the design and development of a product in a certain domain. It defines the requirements that a product must follow and the properties that it should adhere to for a risk free operation of the product in its environment. It also defines strategies for the mitigation of harm in case of a hazard.

To ensure that a product follows some standard, the requirements of the standard need to be incorporated in product development in the very initial design phase and must be followed throughout the product development lifecycle. For this purpose it is important that the product supplier ensures that the people associated with product development lifecycle must be well aware of the guidelines of the standard and their practices must be aligned with the processes mentioned in the standard.

It is necessary that the product supplier formulates a structured approach for the interpretation of the standard in the context of its products' application domain so that the product designers, developers and testers reach an end product through a systematic process and have enough evidence that supports the compliance of the product with the standard. This evidence that relates the concepts of the standard to the processes of the application domain also provides strong documented proof that helps streamline the process of certification for the certification body to declare conformance of the product with the standard.

## 1.2 Objective

For the certification of functional safety of the E/E systems in road vehicles, the automotive industry is provided with ISO 26262 as a safety standard. ISO 26262 stretches up to about 450 pages and is divided into 10 sections, each stating the requirements for product development, product safety and guidelines for compliance with ISO 26262. These associated requirements are grouped together and termed as Work Products. Compliance with the standard therefore means production of a system in agreement with these work products.

For a E/E system or component to comply with ISO 26262, it should be developed and tested in view of the work products defined in the standard. Developing a product according to guidelines mentioned in the standard and verifying that it is in compliance with the standard can become a challenging task considering the fact that the standard is a textual document and is subject to individual interpretation. Also, it is impractical to go through the standard over and over again to check if all the requirements are being met. For this reason, companies adopt development processes keeping in mind the requirements of the standard that, if followed correctly, result in a product that conforms with the standards. It is therefore important to make sure that following the development processes results in a product compliant with the

standard, and that a given development is following the company processes. Both of these are difficult and time consuming tasks.

Although the requirements and guidelines are clearly mentioned for each work product, without an explicit understanding of these requirements, the product under development is prone to deviation from the standard. This would result in a product not compliant with the standard and would require re-design and re-development of the product which is a time consuming process and also becomes expensive because of the overutilization of resources. Missing evidence of compliance leads to difficulty in the certification process of the product as well. Therefore, it is necessary to come up with a systematic approach to demonstrate the implicit contents of ISO 26262 explicitly and devise a solution that would serve as a process that can be followed by companies, and would automate the process of checking compliance with the standard, thus providing strong evidence of compliance and reducing the time and resources required for verification and certification.

The aim of this thesis is to present an approach for product design and development in the light of ISO 26262 that helps verify the product against the standard, while reducing the use of resources and time. We use the idea of conceptual modeling in Model Driven Engineering (Brambilla, Cabot, & Wimmer, 2012) to relate the theories of the standard mentioned in textual form to visual diagrams. We implement this approach on a real world product as an example to illustrate the use of our idea to check compliance of a product with ISO 26262 using conceptual modeling.

### **1.3 Related Work**

Due to their importance in the industry, much work has been done to help implement the safety standards in practice. Modeling of safety standards is one of the techniques that researchers follow to come up with ideas to help develop and implement a stan-

standardized process for compliance checking of a product against a safety standard. The authors of (Panesar-Walawege, Sabetzadeh, & Briand, 2013) use MDE techniques to show an idea of certification with safety standards through conceptual modeling and provide a framework for compliance for IEC 61508. They develop domain models of the safety critical systems and evidence models of IEC 61508 and demonstrate relationships between the two to check compliance. The conceptual models are created by carefully analyzing the text of the standard to identify the important concepts mentioned in the standard. These concepts are then represented as classes in UML (*OMG Unified Modeling Language<sup>TM</sup>, 2015. v2.5., n.d.*) class diagrams and are related to other classes according to the relationships defined between them in the standard. The UML models are then extended by UML profiles by introducing context-specific stereotypes. The UML profiles are augmented with OCL constraints to help implement the checks and restrictions on the underlying UML class diagrams. We have used similar technique of using models of standards to show compliance of a product with the standard. As ISO 26262 is an adaptation of IEC 61508 and has a different structure and requirements, the process that we adopted for building conceptual models of ISO 26262 is different from what has been done for IEC 61508. ISO 26262 groups together associated requirements into Work Products so to comply with the standard, a product has to comply with the requirements of every work product in the standard. This procedure is demonstrated in the following chapters of the thesis. Creation of UML profiles to apply OCL constraints is an overhead while creating models of the standard. In our thesis, we have omitted the use of UML profiles thus simplifying the use of modeling to check compliance of a product with ISO 26262.

Although (Luo et al., 2013) uses a similar technique, they use the snowball methodology to create conceptual models and process models of ISO 26262. The requirements of the standard are classified as high level and low level requirements for the creation of conceptual models as UML class diagrams. First, a basic model is created from the high level requirements. Just like a snow ball gets bigger as it is rolled, the conceptual model becomes more elaborate by gradually adding the high level and then the low level requirements. Next, the process models are created and represented as activity diagrams and the work products are assigned to relevant activities, tasks or phases.

For validation, these models are compared to industrial models that are used to check compliance with ISO 26262. The approach used in this paper, although valuable, does not take into account the constraints mentioned in the standard. Although it talks about model verification and validation, there is no process defined to check the compliance of a sample product with the models of the standard. In our thesis, we have created conceptual models demonstrating the core concepts of the standard as well as the constraints on these concepts. Also, we have presented a process of checking compliance of an industrial product with ISO 26262.

## 1.4 Organization of the thesis

The second chapter talks about background knowledge of ISO 26262 and provides an insight into its contents and work flow. It also talks about the core concepts that are to be used to demonstrate our idea of checking compliance with ISO 26262. The third chapter gives a detailed description of the Fuel Level Estimation and Display System (FLEDS) (Dardar, 2014). We use FLEDS as an example to illustrate our idea of compliance checking. It also presents the conceptual models built from the requirements of some of the work products and explains the idea of measuring compliance with the standard using those conceptual models. Chapter 4 gives an example of how the conceptual models can be used to check compliance. Chapter 5 highlights the challenges and limitations faced during this work and finally gives a conclusion of our thesis.

# Chapter 2

## Basic Concepts of ISO 26262

### 2.1 About the Standard

ISO 26262 is an adaptation of IEC 61508 which is an international standard for electrical, electronic and programmable electronic safety related systems. Published by the International Electrotechnical Commission, IEC 61508 is a basic functional safety standard applicable to all kinds of industry. ISO 26262 is designed for series production passenger cars with one or more E/E systems and an average vehicle mass up to 3500 kg. It applies to the activities involved in the safety lifecycle of safety-related systems comprised of electrical, electronic and software components in road vehicles.

ISO 26262 is a functional safety standard and it addresses potential hazards that could be caused by the malfunctioning of E/E systems. Just like IEC 61508 uses Safety Integrity Levels (SIL), ISO 26262 also uses Automotive Safety Integrity Levels (ASIL). ASIL classification is a risk based approach to determine the reliability of a

system or to express the level of risk reduction required to prevent a specific hazard. It helps define the safety requirements necessary to be met for a system to be deemed safe in light of ISO 26262. The ASIL levels are assigned through risk analysis techniques by identifying the hazards associated with the system and assigning ASILs to them.

## 2.2 Work Product

For a system to be functionally safe according to the standard, it should comply with the requirements and guidelines set forth by the standard combined together in what are called work products. According to the vocabulary section of the standard, a Work Product is a

*“result of one or more associated requirements of ISO 26262”*

Therefore, for an E/E system to observe the guidelines and requirements defined in the standard, it should comply with the work products in the standard.

## 2.3 Work Flow

The standard stretches up to about 450 pages divided into 10 parts. Each part lists work products, which are the results of requirements and recommendations for the design and development of functionally safe E/E systems. There are about 120 work products mentioned in the standard so, for a system to comply with the standard, it needs to comply with all the requirements that result into these work products.

Figure 2.1 shows the overall structure of ISO 26262. ISO 26262 uses a V-model, also known as a verification and validation model, for the product development phases. The shaded region in the diagram shows the interconnection of different parts of the standard in a sequence. The standard has a total of ten parts. The first part defines the terms used in the standard and their relationship with other terms. The second part defines the safety activities or requirements that are critical in the safety lifecycle of a system. The third part gives an understanding of the development of a system, the hazards associated with the system and the functional safety concept of the system. It gives insight into the elements that form the structure of a system, its boundaries, the classification of elements, possible hazards and their classification and guidelines for hazard mitigation. Overall, it gives an overview of the properties of ISO 26262 and helps acquire familiarity with the standard. So, for the purpose of our thesis, we focus mainly on Part 3 of the standard which is the Concept Phase (ISO 26262-3, 2011).

Figure 2.2 shows a general work flow of the safety lifecycle of a system as defined in Part 3 of ISO 26262. Based on this work flow, we pick the three main work products defined in Part 3 of the standard to demonstrate our idea of compliance with ISO 26262. These work products are further talked about in the remaining parts of this thesis.



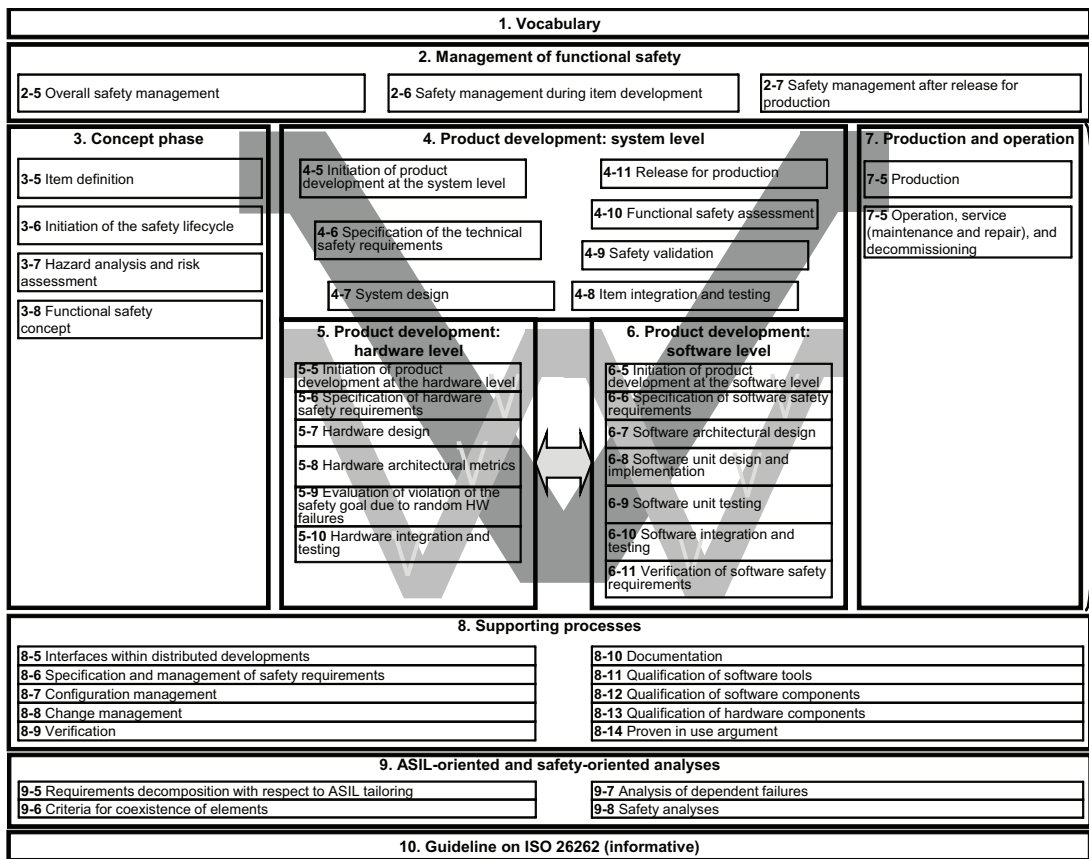
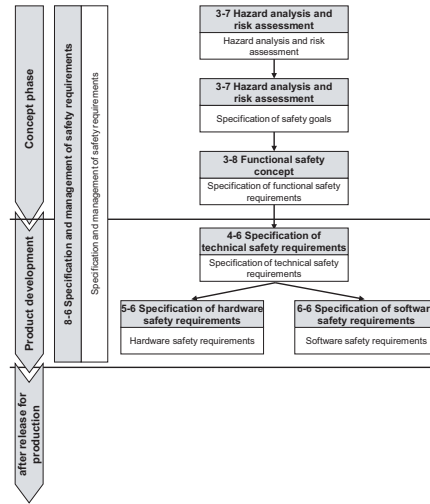


Figure 2.1: Overview of ISO 26262 (ISO 26262, 2011)



NOTE Within the figure, the specific clauses of each part of ISO 26262 are indicated in the following manner: "m-n", where "m" represents the number of the part and "n" indicates the number of the clause, e.g. "3-6" represents Clause 6 of ISO 26262-3.

Figure 2.2: Structure of the safety requirements (ISO 26262, 2011)

## 2.4 Conceptual Modeling

As explained earlier, the requirements of ISO 26262 are grouped together in work products. So, for compliance with ISO 26262 it is necessary for a product to be in conformance with the work products of ISO 26262. In this thesis, we will explore a possibility to verify compliance of a product with ISO 26262 by relating the products' properties with the requirements of the work products. We use the approach of conceptual modeling (Pastor & Molina, 2007) of a system for compliance checking with ISO 26262.

A conceptual model, also known as a domain model is a descriptive model of a system representing its core ideas and concepts. The model captures the essence of the system by presenting it in terms of its elements, their core features and relationships with

other elements and their boundaries. The conceptual model illustrates a schema of the system that can be represented using a descriptive language or diagram. A conceptual model helps in establishing the concepts, defining the scope and providing a base model for future development processes. The conceptual model of a system increases the understanding of the domain for the users and developers of the system.

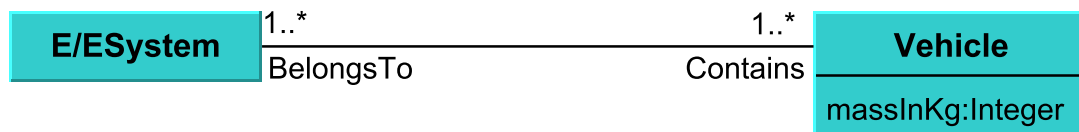
We use the idea of conceptual modeling to demonstrate the contents of ISO 26262 in a visual form, thereby reducing ambiguities that may arise due to the natural language of the standard and providing a process that could be used in future as a basis for the design and development of products in compliance with ISO 26262. In the context of this thesis, we call a conceptual model a model resulting from a conceptualization or abstraction process. Since a conceptual model is a model of a system, the system in our case is ISO 26262, and more precisely, its work products. In other words, we will be building the conceptual models of the work products of ISO 26262 to enable us to represent the main elements appearing in these work products so as to being able to perform analyses. This exercise is not easy, as it highlights what the standard says, what it doesn't explicitly convey and what it lacks.

Different conceptual modeling techniques are used for different purposes. For example, Data Flow modeling is a technique in conceptual modeling used to give a graphical overview of the flow of data through a system. An Entity-Relationship model is used for the design of relational databases in a system and represents the abstract representation of a data model. We adopt an object oriented approach to demonstrate the fundamental ideas and requirements of ISO 26262 work products and the relationships between them. Since ER diagrams are used for mapping onto the tables in a database, we use class diagrams to represent our object oriented models. Class diagrams are widely used for modeling object oriented systems. They capture the fundamental concepts in a system by representing them as classes, their attributes and the relationship between those classes. Class diagrams support the concepts of object oriented design such as inheritance, associations, compositions and aggregations. In addition, class diagrams have also been used as a tool for supporting compliance checking with safety standards (Panesar-Walawege et al., 2013). We use Unified Mod-

UML (Unified Modeling Language) (Li, 2007) as a modeling notation, and more precisely, as a language for class diagrams. We use UML class diagrams based on the fact that they are well-organized, have a wide tool support for creating object oriented design and code skeleton and are easy to learn for desired communication between product analysts, architects, developers, stakeholders and users.

UML diagrams represent the structural and behavioral view of a system. The structural view is static and revolves around objects in a system, their attributes, their relationships with other objects and the operations they perform. This view is represented using a class diagram. The behavioral view is dynamic and shows the state changes, activities and collaborations of objects in a system through diagrams such as sequence diagrams and activity diagrams. To exhibit the structural view of ISO 26262, we make use of UML class diagrams.

In class diagrams, entities or objects are represented as classes. Each class is represented as a box that has its name, attributes and operations listed. The classes are connected to other classes through different arrows that show the relationships between those classes. The relationship could be an association, dependency, inheritance, aggregation or a composition. Multiplicity constraints in UML allow specification of cardinalities which help in grouping together elements. For example, ISO 26262 talks about E/E systems which belong to vehicles having a certain mass. This idea is represented in a class diagram as



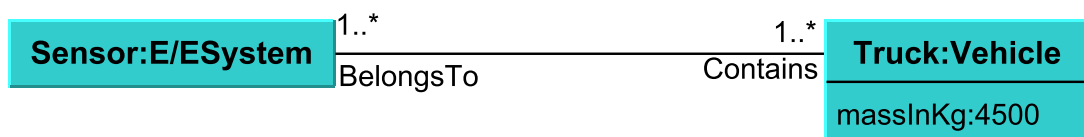
From this class diagram we know that a vehicle with some mass has one or more E/E

systems and an E/E system can belong to one or more vehicle.

The conceptual modeling in UML can serve as a basis for creating meta-models for software development of the domain and can also be converted to other modeling languages. This makes conceptual modeling using UML highly adaptable and convenient for use in industrial processes and hence we choose it as a modeling technique for ISO 26262 conceptual models.

## 2.5 Object Diagram

Let us assume that we have a conceptual model of each of the ISO 26262 work products. If we wish to address compliance with the standard it is important to distinguish an application of the standard from the standard itself, as they are at different levels of discourse. In the context of conceptual models of ISO 26262 work products, an application of the standard is a relation between a conceptual model and a particular instance of that conceptual model. Since we have chosen UML as a modeling notation for class diagrams, we can capture this relation precisely as a conformance relation between a class diagram and an object diagram. An object diagram is a visual representation of an instance of a class diagram, including objects of the classes of a class diagram and data values of the attributes of the classes. An example object diagram of the sample class diagram mentioned earlier can be



This instance of the class diagram shows that a truck of mass 4500 kg has a sensor as one of its E/E components.

## 2.6 Constraint Language

Expressing the ISO work products as UML diagrams provides a graphical layout of the structure of the work products, but the UML diagrams do not say anything in particular about the guidelines or restrictions that are vital to these work products. The relational constraints of objects in a class diagrams can be defined using multiplicity in UML, but the constraints that ISO 26262 defines for the properties of objects cannot be represented diagrammatically.

To incorporate the constraints in a system's domain and to demonstrate the precise specifications, UML provides a constraint language called the Object Constraint Language (OCL) (*OMG. Object Constraint Language, 2014. v2.4., n.d.*). OCL is a declarative language that helps in overcoming the limitations of UML by allowing specification of logical properties of object-oriented models. OCL thus helps in describing the rules applying to the UML models. Developed by IBM in 1995 and originally used as a query language for business modeling, OCL is now a part of the official Object Management Group (OMG) standard for UML.

The selection of UML and OCL as modeling languages for conceptual modeling of ISO 26262 work products is a design choice we have made. There are other modeling languages that can be used for design specification but because UML is a fairly old, easy to learn and widely accepted language, the tool support for UML is more and hence is more commonly used in the industry. In the sample class diagrams mentioned in the previous sections, we have mentioned a truck of mass 4500 kg. We know that ISO 26262 requires that the vehicles' mass be up to 3500 kg. This is a property that cannot be verified using UML relationships and cardinality constraints. Therefore,

we define an OCL constraint for our class diagram that checks if the object is in conformance with the properties defined in the class diagrams. The following piece of code shows how this constraint is defined using OCL:

```
context vehicle
inv vehicleMass:
massInKg <= 3500
```

## 2.7 Selection of Tools

There is a wide variety of tools available to create UML class diagrams. The selection of a tool however, is to be based on its ability to support OCL as a constraint language. We require a tool that helps us create conceptual models using UML which can be augmented with OCL to define the system constraints. The tool should also allow creation of model instances and the verification of those instances against their models. The tool should check the structural requirements as well as the OCL constraints that are required of an object model of a class diagram. The Eclipse Modeling Framework (EMF) provides a wide variety of model development tools and plugins that help define UML models with OCL constraints. These tools require the creation of a UML profile to define OCL constraints before the definition of UML class diagrams. The instances of these class diagrams are then checked against the class diagrams as well as UML profiles. It is to be kept in mind that although an instance specification of a class diagram is supported by EMF, the documentation of some eclipse modeling frameworks states that they do not provide support for verification of instance specification against the UML models and only provide the model verification ability through UML profiles. The creation of UML profiles just for the definition of constraints seems to be an unnecessary overhead for the aim of this thesis. Therefore, we look for simpler tools that ease the task of model specification.

The USE (UML-based Specification Environment) tool (University, 2007; Buttner,

& Richters., 2007) is a model definition tool based on the textual description of a model using the basic features of UML. This text based model definition allows the specification of OCL constraints while defining the classes and their associations. The textual model, that is referred to as a USE specification, in the documentation of the tool, can then also be graphically represented and verified against the OCL constraints by using this tool. The USE tool also allows instantiation of a class diagram by creation of object diagrams. These object diagrams can be checked for compliance against their respective class diagrams by evaluating the OCL constraints defined in the textual models of class diagrams. This tool also checks the multiplicity constraints, which is a useful feature to determine the intactness of the structure of the model. Therefore, we use this tool to define class diagrams, constraints on these class diagrams, instances of these class diagrams and to verify that these instances are in agreement with their respective class diagrams. There are plenty of other tools that support OCL constraints for UML class diagrams, but the model verification ability of USE is the property that stands out in tool selection. USE also allows the generation of Java code for the models which make it an ideal choice since it is convenient to translate from Java to any other modeling technology or even create meta-models for future software development. Table 2.1 shows a comparison of the two frameworks by listing the steps required for model verification:

	<b>EMF</b>	<b>USE</b>
<b>1.</b>	Create UML profile with stereotype : automobile	Create UML class diagram with class vehicle
<b>2.</b>	Create OCL constraint for automobile	Create OCL constraint for Vehicle
<b>3.</b>	Create class diagram with class vehicle having classifier automobile	Create Instance Diagram
<b>4.</b>	Create Instance of class diagram	Validate Instance
<b>5.</b>	Validate Instance	

Table 2.1: Comparison of EMF and USE Framework



One drawback of USE is its inability to create presentable and tidy class diagrams. Although the tool provides features to automatically align the classes, but because the conceptual models have a large number of classes, it is vital that we represent an immaculate diagram for a conceptual model to aid visualization by the user. So instead of using USE's display feature to show class diagrams we use yEd Graph Editor (n.d., n.d.). yEd is a tool provided by yWorks for the creation of high quality diagrams for software development and documentation. yEd provides excellent user environment to define the classes of a UML diagram and the relationships between them. It provides automatic rearranging of the layout of diagrams which helps in creating immaculate designs that are easy to follow.

In USE specification, the example class diagram, OCL constraint and object diagram are represented using the following code:

```
model ISO26262

class vehicle
attributes
massInKg : Integer
end

class EESystem
end

association VehicleconstiansSystem between
vehicle[1..*]
EESystem[1..*]
end

constraints

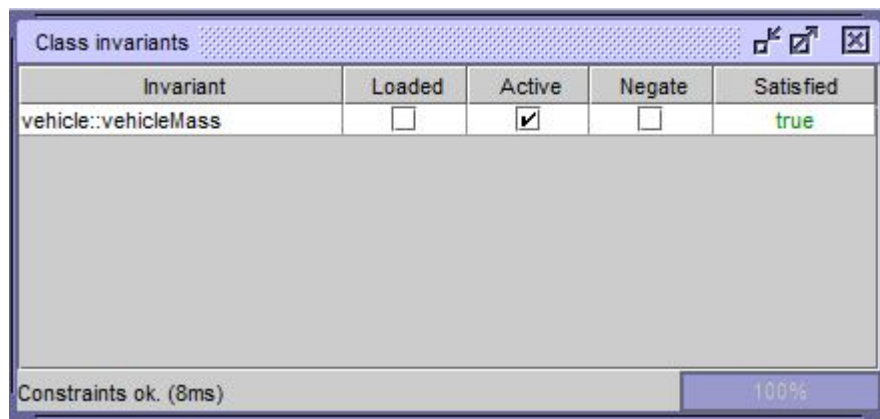
context vehicle
--Mass of vehicle should be up to 3500 kg
```

```
inv vehicleMass:
massInKg <= 3500

*****
--Object Diagram

!create truck : vehicle
!create sensor : EESystem
!set truck.massInKg := 3500
!insert (truck,sensor) into VehicleconstiansSystem
```

Figure 2.3 shows that the OCL constraint for the example model is verified in USE.



Invariant	Loaded	Active	Negate	Satisfied
vehicle::vehicleMass	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	true

Constraints ok. (8ms) 100%

Figure 2.3: OCL Verification in USE

Figure 2.4 shows the log for the structure of our model which is valid as all the multiplicity constraints are satisfied.

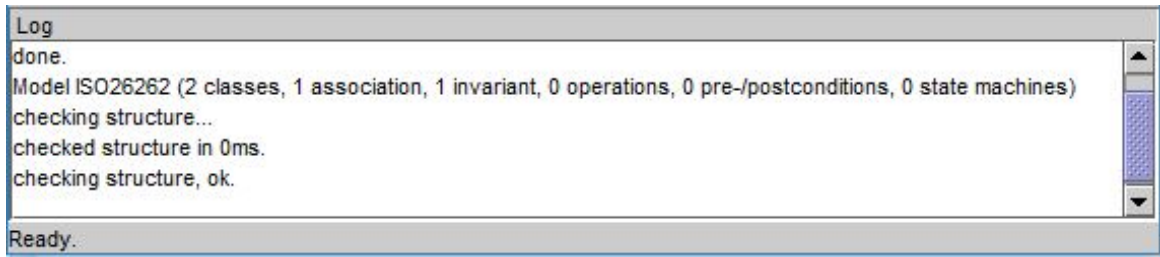


Figure 2.4: Structure Verification in USE

# Chapter 3

## Building the Conceptual Models

### 3.1 Approach

Now that we have described the use of conceptual modeling and OCL constraints in Model Driven Engineering (Luo, van den Brand, Engelen, & Klabbers, 2014; Gogolla, 2011), we will implement our idea using these techniques. We aim to describe the natural language of ISO 26262 in a formal language notation using UML to present a structured overview of the contents of the standard, hence reducing the ambiguities that may arise while interpreting the textual documentation. The essence of ISO 26262, as mentioned earlier, is in the work products that result from the requirements and recommendations highlighted in the standard. Therefore, we make use of these requirements and recommendations to identify the concepts of a work product and come up with an approach to establish conceptual model for work products. These concepts have their roots in the vocabulary section of the standard that helps further elaborate the relationship between individual terms mentioned in the work products.

For a proof of concept, we create UML class diagrams annotated with OCL constraints of three of the work products of ISO 26262, representing the core content of the work products as classes, attributes and the relationships between other classes and their attributes. These UML class diagrams are defined using USE specifications. Then we augment these models with OCL constraints to incorporate the rules and restrictions mentioned in the work products to achieve a complete conceptual model. Lastly, we use an example system to create instances of the conceptual models and check the instances against their conceptual models to verify if the system is in compliance with ISO 26262.

## 3.2 Selection of Work Products

To illustrate our idea, we choose three work products as a proof of concept. Starting from section 2, each section of ISO 26262 has several work products defined. We choose the following three work products from section 3 of the standard:

- Item Definition
- Hazard Analysis and Risk Assessment
- Safety Goals

The reason for choosing these three work products to demonstrate our idea is that they give an overview of the content of the standard and also give an outline of the set of requirements of a products safety lifecycle. Work product Item Definition defines how the structure of an item should be categorized, how its relationship with other items and its environment should be described and the conditions that should be defined leading the item to perform its functionalities. Hazard Analysis and Risk

Assessment talk about what a hazard is, how it is related to an item, the causes of a hazard and the parameters that need to be considered while classifying a hazard. Work product Safety Goals then gives requirements that need to be adhered to while defining a safety goal for an item and also describes the relationship of a safety goal with the hazards associated with the item. In short, the three work products give a picture of the series of phases covering design and development of safety features of a system in accordance with ISO 26262.

### **3.3 Creation of Class Diagrams**

Each of the work products defined in ISO 26262 has a set of requirement and recommendations that need to be fulfilled by a product for it to be in coherence with the work product. The requirement analysis is a preliminary step in this regard. The requirements defined are in an informal and natural language which has inherent ambiguity and uncertainty. To define the conceptual models for work products and represent them as class diagrams, we adopt a standard approach for extracting information from their requirements. The requirements are defined in sentences which are a combination of nouns, noun phrases, verbs and verb phrases. The nouns in the requirements are classified as classes and the properties defined for them are categorized as the attributes of these classes. The verbs and verb phrases help us identify the relationships between these classes which we classify as associations, aggregations, compositions and dependencies. We see that the requirements sometimes define a hierarchical relationship between these classes, which we represent using generalization relations. Some of the relationships also define cardinalities which we define as multiplicity constraints like one-to-one, one-to-many or many-to-many relationships. There are some restrictions that are applicable to these classes which cannot be defined as relationships or cardinality constraints so, to represent such features, we use OCL constraints. This approach will be further explained as we go on to define the conceptual models of the three aforementioned work products in the following sections.

### 3.4 Item Definition

Work product Item Definition provides the requirements for the definition of an item and guidelines to describe its dependencies on other items or its interaction with its environment. This work product requires us to define the functionality of the item, its operating conditions, legal requirements and associated hazards. Since this work product is the first in the definition of the safety life cycle of an E/E system, it has no prerequisites. Item Definition is a result of the requirements and recommendations of Section 5.4 of Part 3 of the standard. Figure 3.1 shows an excerpt of what these requirements and recommendations are presented in the standard.

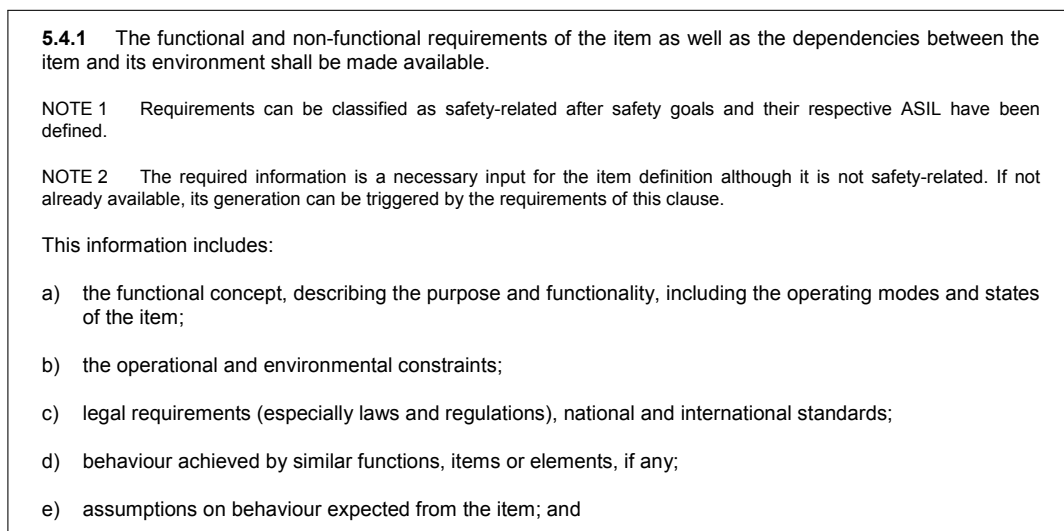


Figure 3.1: Requirements and Recommendations for the *Item Definition* (ISO 26262, 2011)

The standard also specifies that an item interacts with other items, its interfaces and boundaries. Figure 3.2 shows an excerpt of how this information is presented in the standard.

Since the work product revolves around the term Item, the first class we define in our conceptual model is the class Item. Now, to define the relationship of item with other classes, we look at the requirements of the work product. From the requirements

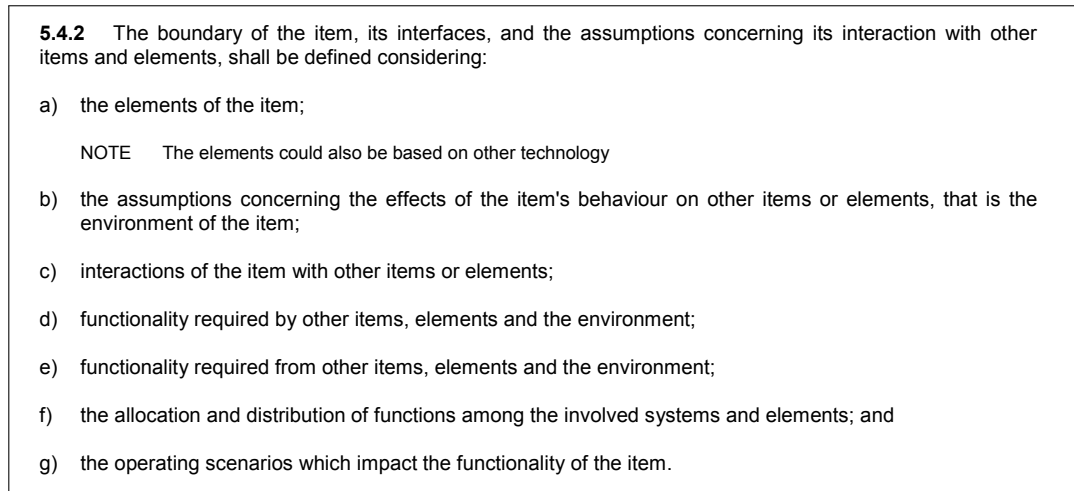


Figure 3.2: Requirements and Recommendations for the *Item Definition* (ISO 26262, 2011)

mentioned above we see that an item has some functionality, behavior, constraints and requirements. It also suggests that the functionality of an item can have effects on other items in its environment and vice versa, it can be affected by the behavior of other items or elements in its environment. To represent these requirements as class diagrams, we highlight the nouns and their relationships with other nouns. For example from the first requirement which is

*“the functional concept, describing the purpose and functionality, including the operating modes and states of the item”*

We highlight the terms *purpose*, *functionality*, *operating modes* and *states*. Before terming them as classes in the conceptual model of Item Definition, we look into the standard to find out how these terms are formally defined. We see that in the vocabulary section, which is the first part of the standard, the term Intended Functionality is defined as the

*“behavior specified for an item (1.69), system (1.129), or element (1.32) excluding safety mechanisms (1.111)”*



This shows that the standard refers to the functionality of an item by the term Intended Functionality. So, instead of defining a class with name functionality, we define a class named Intended Functionality. Similarly, when we look at how the standard defines an operating mode, we see that an operating mode is a

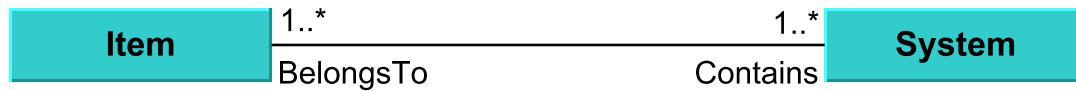
*“perceivable functional state of an item (1.69) or element (1.32)”*

As it is a state of an item required for a certain functionality, we define the operating mode as an association class to elaborate the nature of the relationship between an item and its intended functionality. From the definition we also know that an operating mode is the state of an item, therefore, we do not create a separate state class. Since the purpose of an item is its inherent property, we define it as an attribute of the class item. So from the requirement highlighted above, we get the a class named Intended Functionality and an association class named Operating Mode.

Note that the work product Item Definition only lists the requirements of an item and does not define the term item. We look into the standard to see which entities can be referred to as items in an electrical and/or electronic system according to ISO 26262. For this purpose we refer to part 1 of the standard which is the Vocabulary section (ISO 26262-1, 2011). Looking at the vocabulary part of the standard, we try to decipher the definition of an item to see what it means and what it is composed of. According to clause 1.69 of part 1 of the standard an item is

*“System (1.29) or array of systems to implement a function at the vehicle level, to which ISO 26262 is applied”*

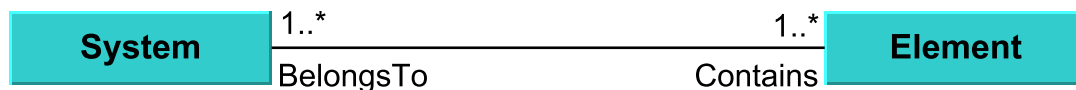
This suggests that an item itself can be called a system or can be composed of more than one system. So, to define an *item*, we create a class *system* and define an association relation between an item and a system as follows:



The multiplicity constraints here mean that an item can contain one or more systems and a system can belong to one or more items. We see that the definition of an Item references the term System. According to ISO 26262 a system is

*“set of elements (1.32) that relates at least a sensor, a controller and an actuator with one another”*

This definition shows the relationship between a system and an element and we define it as



Similarly, we go on to define *element*, *component*, *hardware* and *software component*. A component according to ISO 26262 is a

*“non-system (1.129) level element (1.32) that is logically and technically separable and is comprised of more than one hardware part (1.55) or of one or more software units (1.125)”*

According to the definition, it is a requirement that a component should be comprised

of more than one hardware part, rendering the cardinality of hardware part to be two or more, and of one or more software unit, making its multiplicity one or more.

Since our conceptual model revolves around Item, we see how an item is defined and related with other classes throughout the standard. Section 10 (ISO 26262-10, 2011) of the standard shows a relationship between the terms item, system, component, hardware part and software unit in Figure 3.3. There are a few issues that need to be pointed out here.

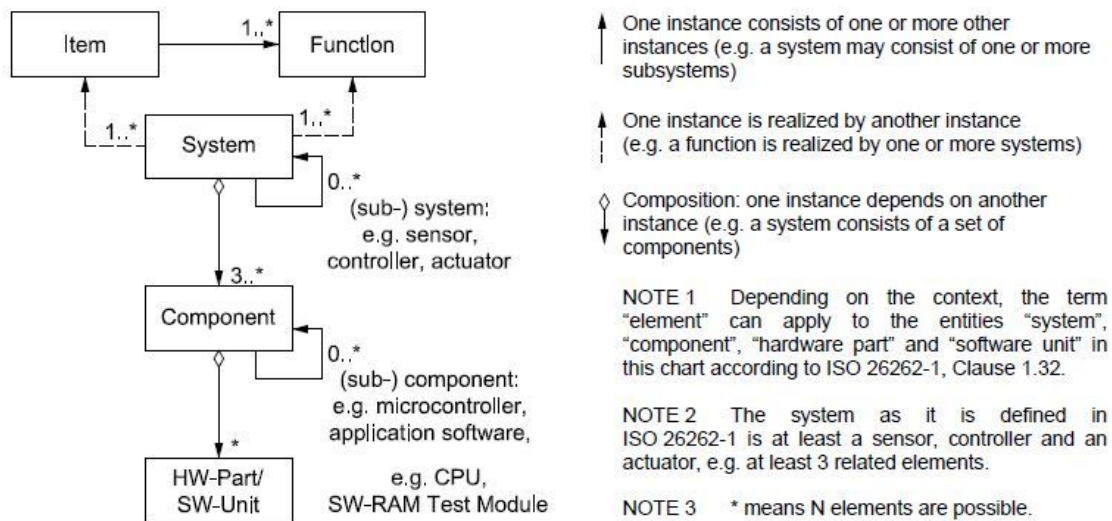


Figure 3.3: Relationship of item, system, component, hardware part and software unit (ISO 26262-10, 2011)

1. From the figure, we see that an array of systems is referred to as subsystem and its multiplicity is defined as zero or more which means that a system can have zero, one or more subsystems related to it. It also exemplifies sensor, controller and actuator as a subsystem. But the definition requires a system to have all three of these elements. The cardinality constraint defined in this case does not

support this requirement.

2. The figure shows that a component has many hardware parts or software units. But, according to the definition of a component, it should have 2 or more hardware parts and one or more software units.

It is to be noted that Element is not included as a separate entity in the model but is only labeled as a system, a subsystem or a component. A system level element is called a subsystem and a non-system level element is referred to as a component.

Keeping these conflicts in mind, we come up with a conceptual model as shown in Figure 3.4 with the intent to define the requirements of the work product Item Definition in a UML model and also to resolve the conflicts mentioned above. We categorize the components as hardware and software components, further classifying them into hardware parts and software units with the required multiplicity constraints.

To define the constraint item puts on a system to have a sensor, controller and actuator, we make use of OCL constraints. To represent it using OCL we write the following piece of code in the USE specification of the model of Item Definition:

```
context Item
inv SensorControllerActuator:
self.relatedSystem->exists (s | s.ocIsKindOf(Sensor)) and
self.relatedSystem->exists (s | s.ocIsKindOf(Controller)) and
self.relatedSystem->exists (s | s.ocIsKindOf(Actuator))
```

Although the constraint applies to a system, we define it in the context of an item since the item itself can be classified as a system by its definition and it is not possible for every system of the item to have a sensor, actuator and a controller. The constraint above checks in all instances of the systems in an item that there should be systems of type sensor, controller and actuator.

Table 3.1 lists the classes and attributes used in the construction of the conceptual model in Figure 3.4 along with their sources from ISO 26262.

### 3.5 Hazard Analysis and Risk Assessment

To illustrate our idea of compliance with the standard, the second work product that we will be building a conceptual model of is the Hazard Analysis and Risk Assessment achieved as a result of the requirements of 7.4.1.1 to 7.4.4.2 mentioned in the Concept Phase of ISO 26262. This work product has Item Definition as a pre requisite work product as the hazards determined in HARA are related to the item resulting from the Item Definition. The objective of HARA is to analyze the hazards caused by malfunctions in an item, categorize these hazards and determine the safety goals for them so as to avoid harm to the item and the environment that the item is functioning in. Since HARA addresses the hazards related to the item, another pre requisite for the definition of this work product is that it should not take into consideration the safety mechanisms that have already been implemented or are in the process of implementation of the item that would potentially avoid hazards to the item.

Before delving into the requirements and recommendations, we see that the terms Hazard and Hazardous Event are predominantly used throughout the definition of the work product and apparently seem similar. To see what the standard classifies as a hazard and hazardous event and what the difference between the two terms is, we refer to the vocabulary section of ISO 26262. According to the definition, a hazard is

*“potential source of harm (1.56) caused by malfunctioning behavior (1.73) of the item (1.69)”*

<b>Type</b>	<b>Name</b>	<b>Reference</b>
class	<b>Item</b>	ISO26262-3-5
class	<b>System</b>	ISO26262-1-1-1.129
class	<b>Element</b>	ISO26262-1-1.32
class	<b>Component</b>	ISO26262-1-1.15
class	<b>Software Component</b>	ISO26262-1-1.123
class	<b>Hardware Component</b>	specialization
class	<b>Software Unit</b>	ISO26262-1-1.125
class	<b>Hardware Part</b>	ISO26262-1-1.55
class	<b>EE System</b>	ISO26262-1-1.31
class	<b>Actuator</b>	ISO26262-1-1.129
class	<b>Controller</b>	ISO26262-1-1.129
class	<b>Sensor</b>	ISO26262-1-1.129
class	<b>Similar Unit</b>	ISO26262-3-5.4.1-d
class	<b>Standard</b>	specialization
class	<b>National Standard</b>	ISO26262-3-5.4.1-c
class	<b>International Standard</b>	ISO26262-3-5.4.1-c
class	<b>Constraint</b>	specialization
class	<b>Operational Constraint</b>	ISO26262-3-5.4.1-b
class	<b>Environmental Constraint</b>	ISO26262-3-5.4.1-b
class	<b>Requirement</b>	specialization
class	<b>Functional Requirement</b>	specialization
class	<b>Non Functional Requirement</b>	specialization
class	<b>Legal Requirement</b>	ISO26262-3-5.4.1-c
class	<b>Law</b>	ISO26262-3-5.4.1-c
class	<b>Regulation</b>	ISO26262-3-5.4.1-c
class	<b>Functional Safety Requirement</b>	ISO26262-1-1.53
class	<b>Technical Safety Requirement</b>	ISO26262-1-1.133
class	<b>Fault</b>	ISO26262-1-1.42
class	<b>Malfunctioning Behaviour</b>	ISO26262-1-1.73
class	<b>Operating Mode</b>	ISO26262-1-1.81
class	<b>Intended Functionality</b>	ISO26262-1-1.68
class	<b>Hazard</b>	ISO26262-1-1.57
class	<b>Assumption</b>	ISO26262-3-5.4.1-e

Table 3.1: Classes and Attributes of Item Definition

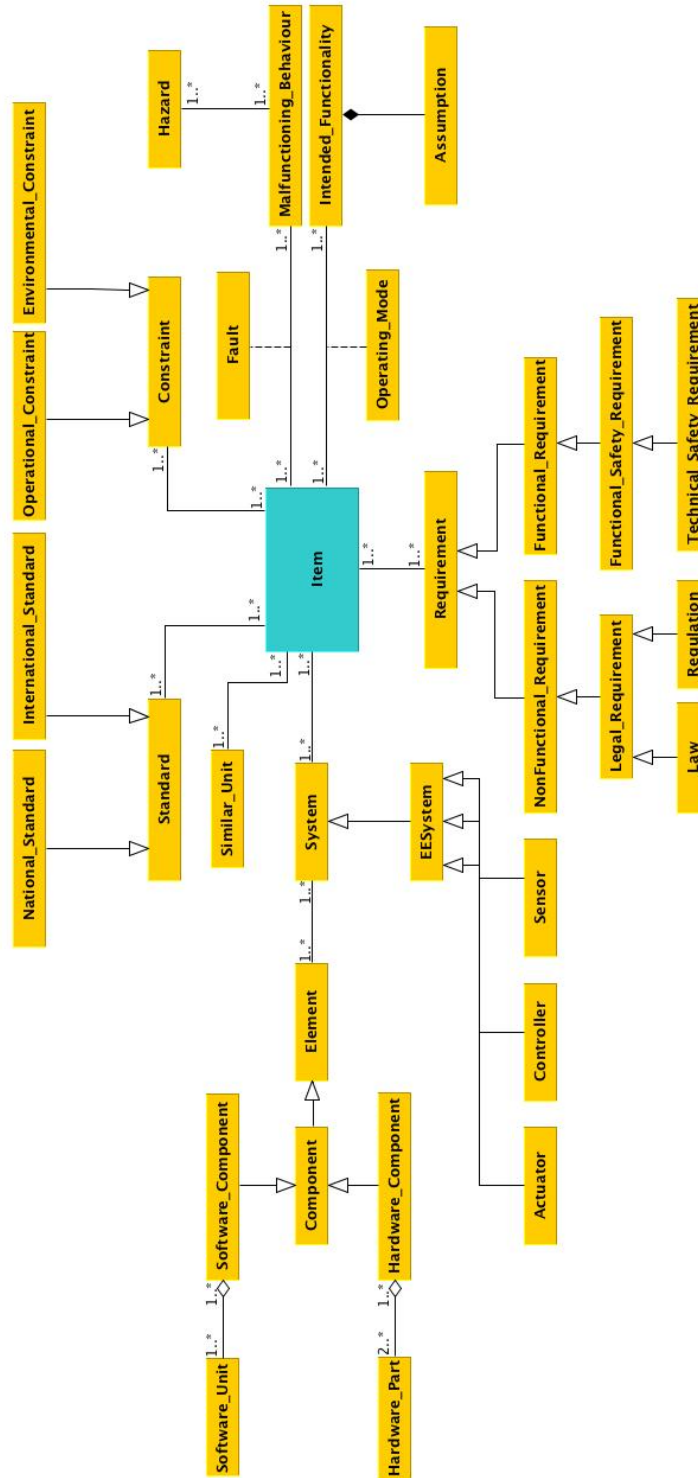
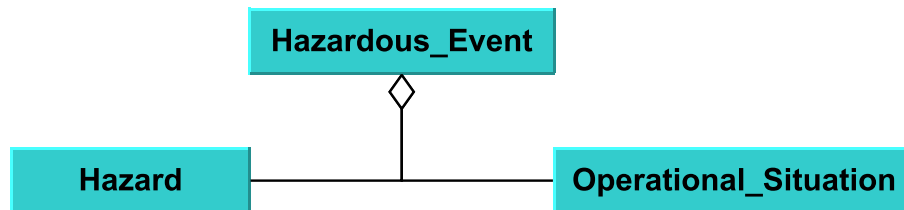


Figure 3.4: Item Definition

and a hazardous event is referred to as

*“combination of a hazard (1.57) and an operational situation (1.83)”*

These definitions help us understand that a hazardous event is caused by a hazard and a particular situation. The term *combination* in the definition helps us define the relationship between the three entities.



Let us now look further into the contents of the requirements and recommendations defined for Hazard Analysis and Risk Assessment. The requirements and recommendations of HARA are divided into the following subsections:

1. Initiation of the hazard analysis and risk assessment
2. Situation analysis and hazard identification
3. Hazard identification
4. Classification of hazardous events
5. Determination of ASIL and safety goals
6. Verification



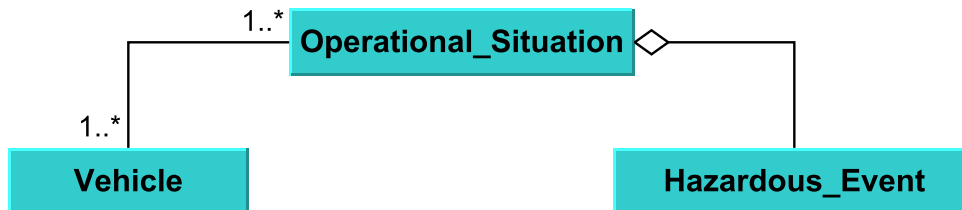
The first clause suggests that the hazard analysis should be performed for the item resulting from work product Item Definition that does not have any safety mechanisms predefined. So we know that a hazardous event is defined for an item; however, we still do not know the relationship between the two terms. The second clause which is the situation analysis gives the following requirement:

*“The operational situations and operating modes in which an item’s malfunctioning behaviour will result in a hazardous event shall be described, both for cases when the vehicle is correctly used and when it is incorrectly used in a foreseeable way.”*

This brings forward three new concepts in the definition of a hazard i.e. *operational situation*, *operating mode* and *malfunctioning behavior*. To find out the meaning of these terms and to define their relationship with an item and its hazards, we again look into the vocabulary section. The term operational situation is defined as a

*“scenario that can occur during a vehicle’s life”*

We had already come across the term operational situation in the definition of a hazardous event. This definition also helps us define a link between a vehicle and a hazardous event.



Through their definitions, we find out that operational situation and operating mode

are two different terms as an operating mode is defined as

*“perceivable functional state of an item (1.69) or element (1.32)”*

This shows that an operational situation is an operating condition of a vehicle whereas the operating mode is the operating condition of an item of a vehicle.

Moving forward, we look into the definition of malfunctioning behavior which is

*“failure (1.39) or unintended behaviour of an item (1.69) with respect to its design intent”*

This brings forward a new term *failure* which is

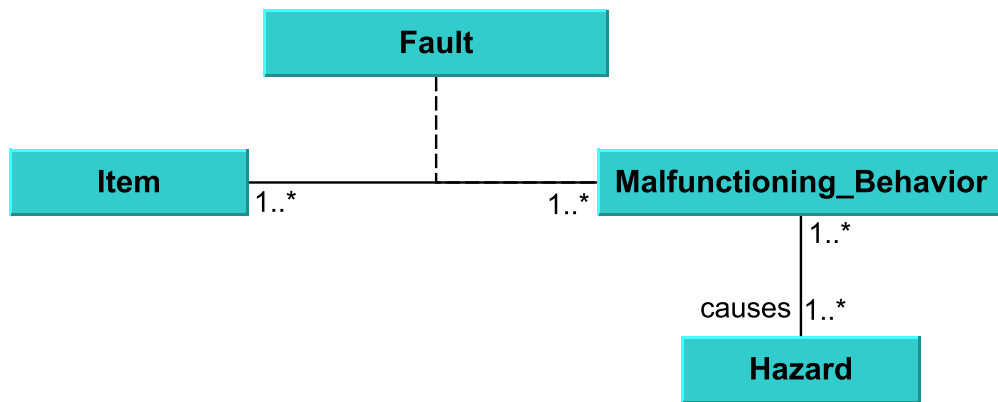
*“termination of the ability of an element (1.32), to perform a function as required”*

While going through the vocabulary section, we also come across the term *fault* which is defined as

*“abnormal condition that can cause an element (1.32) or an item (1.69) to fail”*

From the definitions, the terms malfunctioning behavior, failure and fault seem similar and can cause ambiguity while defining the hazards related to an item. It therefore becomes necessary to differentiate these terms and find out their relationship with an item and with each other. From its definition, we see that a malfunctioning behavior is the failure of an item so the two terms can be interchangeably used. But, from

the definition of fault, we realize that a fault causes a failure. Before defining the relationship between an item, a fault and a failure from the definitions, we see how the relationship is defined between them throughout the standard. In Figure 5 of part 10 of the standard, we see that an example of faults leading to failures is defined. While explaining this example, the standard mentions that a fault can occur at the level of an item as well as at a components level. The fault at a component level will lead to a failure that is a fault at the level of an item, so the failure of a component is an items fault. The fault in an item will lead to a malfunctioning behavior (failure) causing a possible hazard. This helps us define the following relationship:



The third clause from the requirements of HARA states that a hazard analysis technique should be used to determine a hazard and its consequences. The fourth clause introduces the concept of hazard classification using the parameters severity, controllability and probability of exposure. All the hazardous events determined, which are in the scope of ISO 26262, are classified using these three parameters. A hazardous event and the safety goal resulting from the evaluation of hazardous event of an item is given an Automotive Safety Integrity Level (ASIL) using a combination of these attributes.

The severity attribute of the hazardous event is a measure of the severity of harm caused by that hazardous event. A risk analysis is performed to determine the level of

risk each person is subjected to as a result of the hazardous event caused by the item, including the driver or passengers operating the vehicle that might be responsible for the cause of the hazardous event, as well as the pedestrians or the people in other vehicles that might be effected as a result of this hazardous event. ISO 26262 gives a scale of four severity classes. Figure 3.5 shows the severity levels that can be assigned to a hazardous event. In case a hazardous event has severity class S0, no ASIL is assigned.

	<b>Class</b>			
	<b>S0</b>	<b>S1</b>	<b>S2</b>	<b>S3</b>
<b>Description</b>	No injuries	Light and moderate injuries	Severe and life-threatening injuries (survival probable)	Life-threatening injuries (survival uncertain), fatal injuries

Figure 3.5: Severity (ISO 26262-3, 2011)

The probability of exposure is a measure of the probability of the item under consideration being exposed to the operational situation that can cause the associated hazardous event. Figure 3.6 shows the five classes of probability that the standard allows in order of increasing magnitude. The probability however, is not a count of how many vehicles are equipped with the item. If a hazardous event has a probability class of E0, no ASIL assignment is required.

	<b>Class</b>				
	<b>E0</b>	<b>E1</b>	<b>E2</b>	<b>E3</b>	<b>E4</b>
<b>Description</b>	Incredible	Very low probability	Low probability	Medium probability	High probability

Figure 3.6: Probability of Exposure (ISO 26262-3, 2011)

Controllability of a hazardous event is the measure of how much the driver, the passengers or other people potentially at risk are able to take control of the hazardous

event cause by the malfunctioning of the item in order to avoid associated harm. While estimating the controllability, it is assumed that the driver is in an appropriate condition to drive and is observing all the traffic rules and regulations. Class C0 is assigned to a hazardous event if it is controllable in nature and hence does not require ASIL assignment. Figure 3.7 shows the controllability classes defined by ISO 26262.

	Class			
	C0	C1	C2	C3
Description	Controllable in general	Simply controllable	Normally controllable	Difficult to control or uncontrollable

Figure 3.7: Controllability (ISO 26262-3, 2011)

The fourth clause from the requirements and recommendations of HARA gives us a mechanism for the determination of ASIL levels from the three parameters mentioned in the previous clause. After assigning these three parameters to a hazardous event, an ASIL is determined using a combination of the three attributes. Figure 3.8 shows the ASIL levels that should be assigned according to a certain combination of these parameters as described in the standard.

Level A represents the lowest safety integrity level and level D is the highest. In addition to the four ASIL levels, this table also defines class QM (quality management) which symbolizes no requirement to comply with ISO 26262.

Since the class diagram does not provide any assistance with ASIL assignment, we use OCL constraints to determine what ASIL level is to be assigned to the Hazardous Event. For example, the standard says that in case of a probability of exposure of E0, no ASIL is required for the hazardous event. To enforce this requirement on the conceptual model, we write the OCL constraint as

```
context HazardousEvent
inv noASILifE0:
if self.exposure.value = 'E0' then self.asil-> size = 0 else
```

Severity class	Probability class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Figure 3.8: Automotive Safety Integrity Level (ISO 26262-3, 2011)

```
self.asil-> size = 1
endif
```

Also, to determine the right ASIL for the combination of severity, controllability and probability of exposure from the table above, we write an OCL constraint as

```
context HazardousEvent
inv ASILDetermination:
(self.severity.value = 'S1'
and self.exposure.value = 'E4'
and self.controllability.value = 'C2'
implies self.asil.value = 'A')
```

This is one of the possible scenarios from the table for ASIL A. The OCL constraints for other ASIL levels are written in a similar fashion.

Since the Verification sub clause refers to part 8 of the standard, it is outside the scope of the work product HARA and will not be discussed in this section. Table 3.2 list the classes and attributes used in the construction of the conceptual model in Figure 3.9 along with their sources from ISO 26262.

## 3.6 Safety Goals

The third work product that we will be building a conceptual model of in this thesis is Safety Goals resulting from the requirements of 7.4.4.3 to 7.4.4.6. For every hazardous event with an ASIL that is determined through the Hazard Analysis and Risk Assessment, a safety goal is defined. A safety goal is a top level safety requirement that is needed to avoid the risk associated with a hazardous event. A safety goal when achieved ensures that the system is safe enough to operate in a given situation. This safety goal may be a combination of multiple safety goals required to avoid the hazardous event. The work product Safety Goals is achieved as a results of the requirements of 7.4.4.3 to 7.4.4.6 of part 3 of ISO 26262. These requirements are:

1. A safety goal shall be determined for each hazardous event with an ASIL evaluated in the hazard analysis. If similar safety goals are determined, these may be combined into one safety goal.
2. The ASIL determined for the hazardous event shall be assigned to the corresponding safety goal. If similar safety goals are combined into a single one, in accordance with 1, the highest ASIL shall be assigned to the combined safety goal.
3. If a safety goal can be achieved by transitioning to, or by maintaining, one or more safe states, then the corresponding safe state(s) shall be specified.

Type	Name	Reference
class	<b>Hazard</b>	ISO26262-1-1.57
class	<b>Hazardous Event</b>	ISO26262-1-1.59
class	<b>Operational Situation</b>	ISO26262-1-1.83
class	<b>Consequence</b>	ISO26262-3-7.4.2.2.4
class	<b>ASIL</b>	ISO26262-3-7.4.4
class	<b>Severity</b>	ISO26262-3-7.4.3.2
class	<b>Controllability</b>	ISO26262-3-7.4.3.7
class	<b>Exposure</b>	ISO26262-3-7.4.3.4
class	<b>AIS</b>	ISO26262-3-7.4.3.2
class	<b>Harm</b>	ISO26262-1-1.57
class	<b>Severity of Harm</b>	0
class	<b>Probability of Harm</b>	0
class	<b>Risk</b>	0
class	<b>Malfunctioning Behaviour</b>	ISO26262-1-1.129
attribute	<b>Single Point Failure</b>	ISO26262-1-1.121
class	<b>Failure Rate</b>	ISO26262-1-1.41
class	<b>Fault</b>	ISO26262-1-1.42
attribute	<b>Safe Fault</b>	ISO26262-1-1.101
attribute	<b>Transient Fault</b>	ISO26262-1-1.135
attribute	<b>Fault Tolerant Time Interval</b>	ISO26262-1-1.45
attribute	<b>Fault Reaction Time</b>	ISO26262-1-1.44
attribute	<b>Detected Fault</b>	ISO26262-1-1.23
attribute	<b>Perceived Fault</b>	ISO26262-1-1.87
attribute	<b>Permanent Fault</b>	ISO26262-1-1.88
class	<b>Random Hardware Failure</b>	ISO26262-1-1.192
class	<b>Dependent Failure</b>	ISO26262-1-1.1.22
class	<b>Cascading Failure</b>	ISO26262-1-1.13
class	<b>Common Cause Failure</b>	ISO26262-1-1.14
class	<b>Safe State</b>	ISO26262-1-1.102
class	<b>Emergency Operation</b>	ISO26262-1-1.34
attribute	<b>Time Interval</b>	ISO26262-1-1.34

Table 3.2: Classes and Attributes of HARA



4. The safety goals together with their attributes (ASIL) shall be specified in accordance with ISO 26262-8:2011, Clause 6.

From the requirements of Hazard Analysis and Risk Assessment, we know that the hazardous events outside of the scope of ISO 26262 do not need to be assigned an ASIL. Requirement 1 mentioned above restricts that the hazardous events with an ASIL should have safety goals. We write this constraint in OCL as

```
context HazardousEvent
inv ASILthenSafetyGoal:
self.asilLevel->notEmpty implies self.sg->notEmpty
```

Also, it is required that the ASIL level of hazardous event and its safety goal should be the same. We represent this as:

```
context HazardousEvent
inv SameASILValues:
self.asilLevel->notEmpty
and self.sg.asil->notEmpty
implies self.asilLevel.value= self.sg.asil.value
```

After deciding a safety goal, it is further broken down into functional safety requirements. These requirements are further classified as functional safety requirements and technical safety requirements for the elements of the system. Following these safety requirements we can achieve our safety goal hence achieving the safety of our system. The work product Safety Goal results from the requirements of 7.4.4.3 to 7.4.4.6 and results in Figure 3.10.

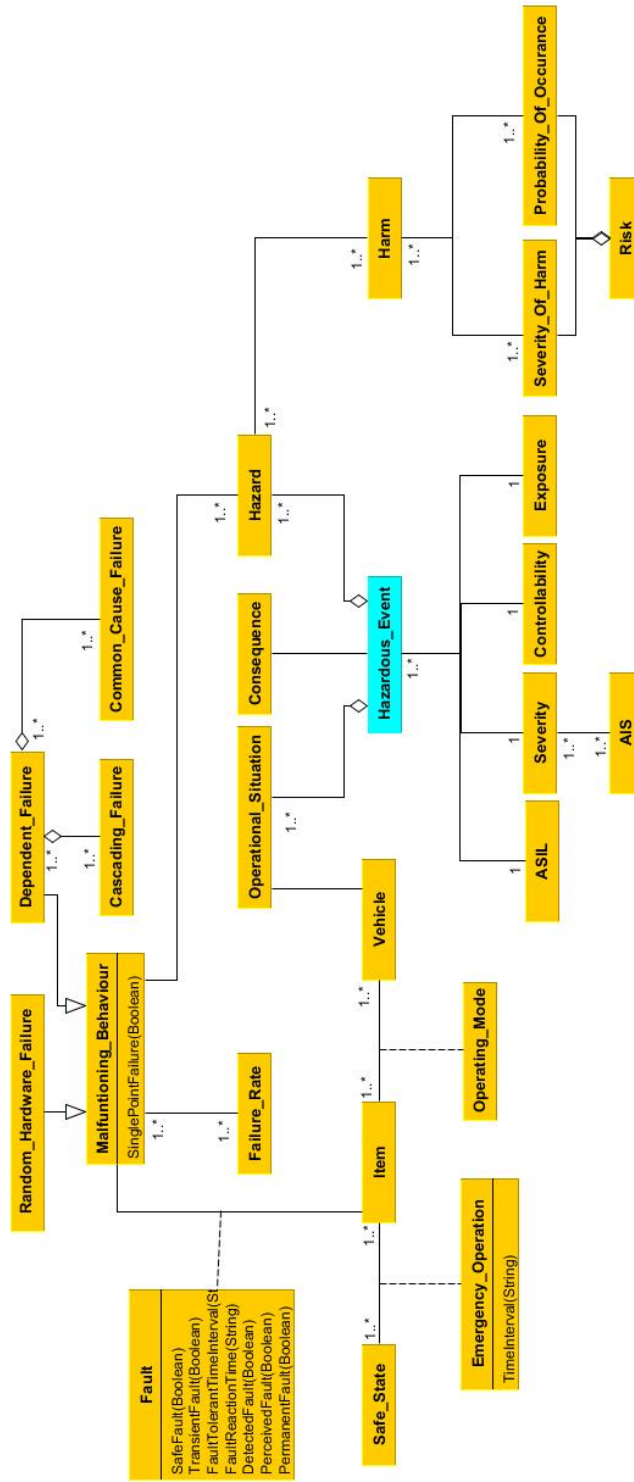


Figure 3.9: Hazard Analysis and Risk Assessment

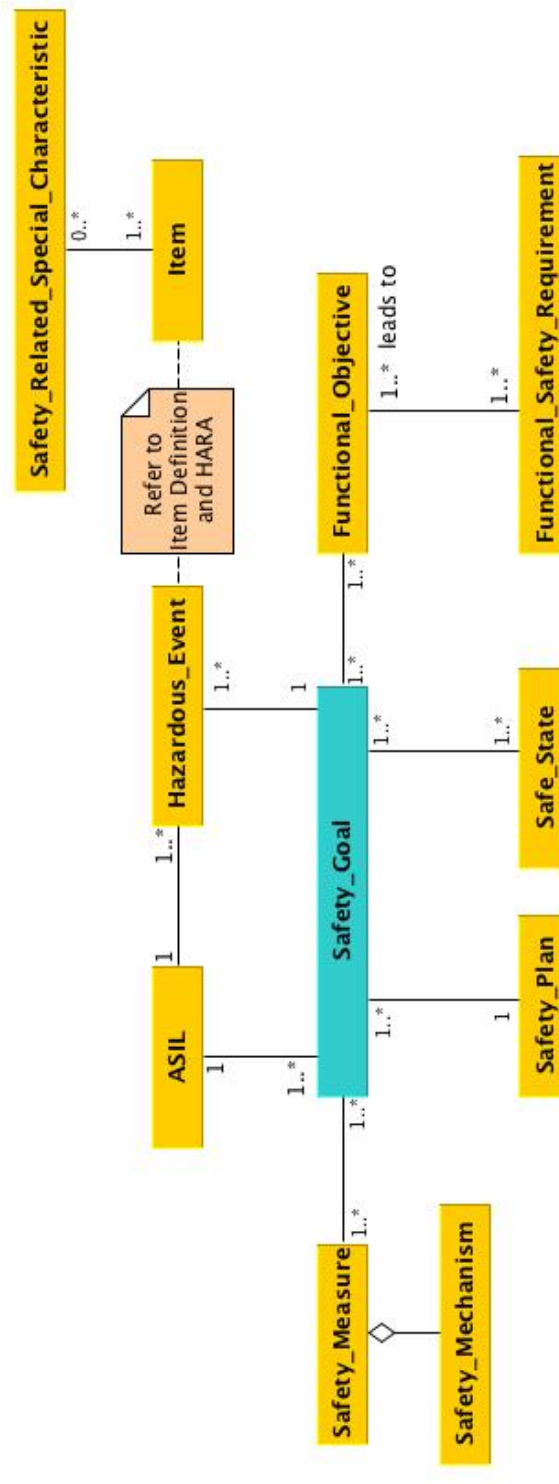


Figure 3.10: Safety Goals

### 3.7 FLEDS - An Example

To explain the idea of compliance of a system with ISO 26262 using a model driven approach, we use Fuel Level Estimation and Display System (Dardar, 2014) as an example. FLEDS is one of the safety-critical systems developed by Scania, a global company that is a leader in sales of trucks and buses. In this section we discuss how FLEDS works, what components it is composed of and what are the related systems required for its operation. FLEDS has the following four variants:

1. Truck with liquid fuel
2. Bus with liquid fuel
3. Truck with gas
4. Bus with gas

We will use truck with liquid fuel as an example variant. Although there are different types of tanks and sensors available, to keep things simple we will consider a truck with one tank and one fuel estimation sensor. FLEDS performs two major functions; fuel level estimation and low fuel level warning. By fuel level estimation, FLEDS provides information to the driver about how much fuel is left in the tank by displaying it on a fuel gauge. This allows the driver to estimate when the tank needs to be refueled. FLEDS also provides a warning to the driver in the case when the fuel level in the tank drops below a certain level. This helps to remind the driver to refuel the tank before completely running out of fuel.

## 3.8 Composition

In Scania, the functions performed by the system that can be experienced by a driver and are based on high level customer requirements are called User Functions (UFs). The user function that implements the working of FLEDS is known as UF18. To implement a UF, one or more Electronic Control Units (ECU) are required which are part of the electrical system of Scania called the Scania Electrical System Architecture for Modularization and Maintenance (SESAMM). Each ECU is composed of one or more hardware and software parts that are needed to perform the activities required by the UF. The hardware parts of the ECUs are made up of sensors and actuators and they are connected together by a Controller Area Network. There are three types of CAN buses used for communication among the ECUs:

1. CAN red: The ECUs connected using the red bus have the highest criticality of 1. These ECUs perform the driveline functions which are responsible for the transmission of power from engine to the wheels and are hence kept separate from other ECUs by connecting through a separate bus.
2. CAN yellow: The ECUs that are next in line to the driveline functions and have the next most criticality of 2 are connected using the yellow bus.
3. CAN green: The remaining ECUs that are less critical having criticality value of 3 are all connected using the green CAN bus.

The ECUs require software to enable the hardware units to perform the required functionality. These software units are pieces of code and are referred to as Allocation Elements (AEs). Each UF is realized by one or more AEs. The FLEDS consists of three ECUs:

1. EMS — Engine Management System

2. COO — Coordinator
3. ICL — Instrument Cluster

The fuel level in the tank is measured using a fuel level sensor placed in the tank. The value calculated by the sensor is sent to the ICL via a CAN bus. The ICL has two components in the dashboard; a fuel gauge and a bulb, for display to the driver. The gauge shows the fuel level and the bulb is used to warn the driver in case of low fuel value. The EMS is responsible for measuring the fuel consumption by the engine to the COO. COO then estimates the fuel level and sends the results to the ICL or in case of low fuel level, issues a warning to the ICL. A battery is used to supply power to the ECUs. Figure 3.11 shows the components of FLEDS.

### **3.9 Allocation Elements**

There are two allocation elements that perform the two basic functions of FLEDS; AE201 and AE202.

1. AE201 is responsible for fuel level estimation. For fuel level estimation, the fuel level sensor placed in the fuel tank is connected to COO and the fuel level is read as a voltage value. The voltage value is transformed into corresponding volume in percentage of the total volume. The current fuel volume is then calculated with reference to the total fuel capacity of the tank. A Kalman Filter Algorithm (KFA) is used to find the fuel level by using the last fuel estimate and the currently calculated fuel level to calculate the new fuel level. The new value calculated is converted into a percentage and sent to ICL to be displayed on the gauge or to AE202 in case the fuel level is below the normal level and a warning needs to be generated.

2. AE202 is responsible for activating a low fuel level warning. The warning is generated based on the current fuel level, the tank capacity and a predefined low fuel level standard for the respective vehicle. After the warning has been generated, the fuel level may increase again for a short period of time because of an uneven terrain, causing the fuel level in the tank to rise above the low level again. To avoid uncertainty in the measurement of the fuel level, the warning is kept for a while. However, if there is a consistent increase in the fuel level or the fuel level changes considerably as in case of a refuel, the warning is removed.

### 3.10 FLEDS Requirements

The high level requirements for UF18 are called the User Function Requirements. These requirements are listed in the technical product documentation provided by Scania for their products. The UFRs as taken from these documents for FLEDS are given in Figure 3.12.

<b>ID</b>	<b>Description</b>
<b>UFR18_1</b>	The indicated fuel level shall not deviate more than $\pm 5\%$ from the actual volume in the fuel tank.
<b>UFR18_4</b>	The low fuel level warning shall warn one time when the estimated fuel level reaches below a limit of the measurable volume in the tank. The limit should be 10% for tank sizes below 900 liters and 7% for larger tanks.
<b>UFR18_8</b>	The estimated fuel level shall be shown immediately when the electrical system is switched on.

Figure 3.12: User Function Requirements

The requirements to implement the allocation elements are called Allocation Element Requirements. The AERs describe how to implement UFRs. The AERs are documented in the technical product data documents published by Scania (1949329, n.d.),

(1949330, n.d.). The AERs for AE201 and AE202 are shown in Figure 3.13 and Figure 3.14 respectively.

The structure and working of FLEDS is explained here in details so that it can be used as an example product to create object models to check compliance with ISO 26262. The structural components, attributes and functionalities are categorized as objects in instance models of the class diagrams of work products discussed in this thesis. These instance models help us verify whether the structure and functionality of FLEDS is defined according to the standard ISO 26262.



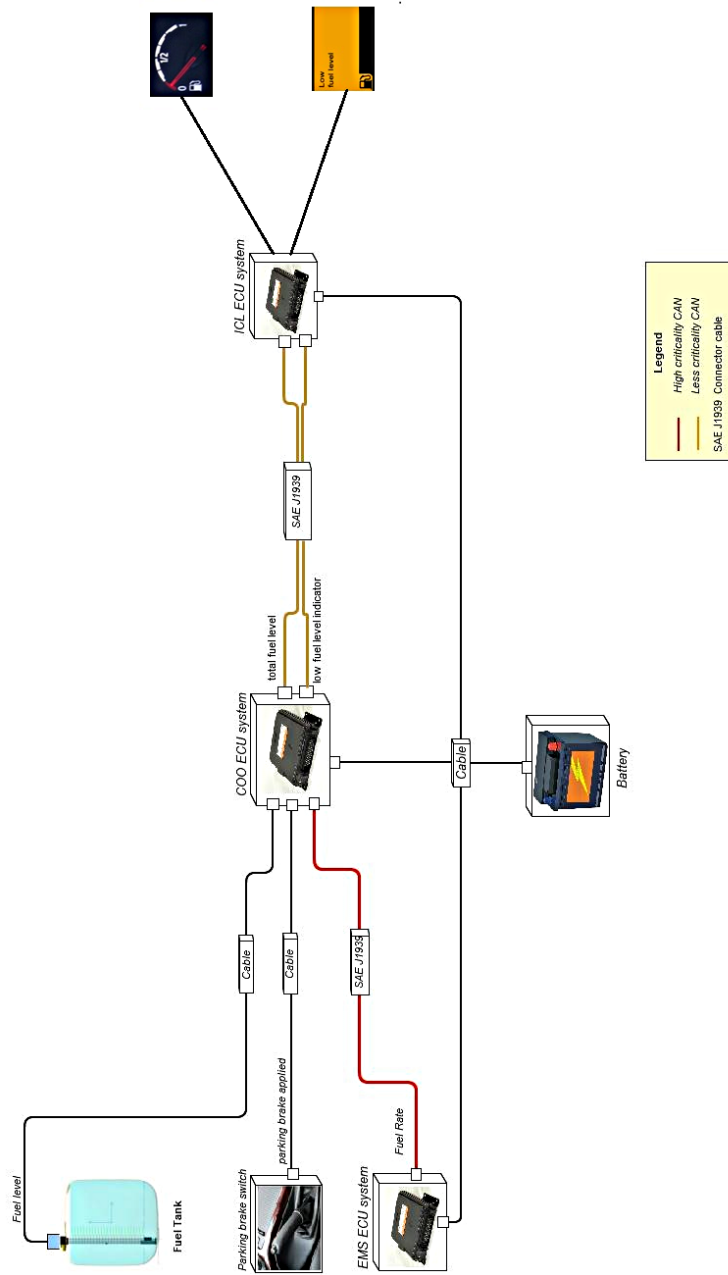


Figure 3.11: Fuel Level Estimation and Display System (Dardar, 2014)

ID	Description
AER_201_9	If the truck has one fuel level sensor connected, its voltage value should be transformed into a corresponding volume in liters. The volume value should be low pass filtered before set to fuelLevel signal.
AER_201_11	<p>TotalFuelLevel should be the output of a filter that includes information from both fuelLevel and fuelRate to achieve a stable signal. The filter should be implemented with a Kalman algorithm as given by the following equation:</p> $x_{start} = \begin{cases} y_s(t), &  y_s - \hat{x}_{old}  > 0.1 X_{tot} \text{ or } y_s > 0.9 X_{tot} \\ \hat{x}_{old}, & \text{otherwise} \end{cases}$ $\hat{x}(t + T_s) = \begin{cases} x_{start}, & \text{during start - up} \\ \hat{x}(t) - T_s u(t) + K(y_s(t) - \hat{x}(t)), & \text{other} \end{cases}$ $y(t) = F(\hat{x}(t))$ <p><b>Where:</b>  <math>y_s</math> = measured fuel level [<math>m^3</math>]  <math>\hat{x}_{old}</math> = fuel level at last shutdown [<math>m^3</math>]  <math>X_{tot}</math> = total fuel volume [<math>m^3</math>]  <math>T_s</math> = sampletime [s]  <math>u</math> = fuel consumption [<math>m^3/s</math>]  <math>\hat{x}</math> = estimated fuel level [<math>m^3</math>]  <math>K</math> = feedback gain [-]  <math>F(x)</math> = function converting <math>m^3</math> to corresponding %  <math>y</math> = total fuel level [%]  <math>x_{start}</math> = start state [<math>m^3</math>]</p>
AER_201_12	The start-up state for the totalFuelLevel estimated should be the state saved from last shutdown if the stored value and fuelLevel doesn't differ with more than 10% of the total volume or it fuelLevel is above 90% of the useable tank capacity.
AER_201_13	If a refill of the tank is done while the ECU is on it should be detected by the algorithm if the sensor(s) indicates a 30% increase compared to the estimated volume. The increase should be held at least 5 seconds so that sloshing is ignored.
AER_201_14	The refill detection should be possible only when the parking brake is applied. The parking brake should be steady applied for at least 5 seconds before the vehicle is considered to be parked.
AER_201_15	If a refill is detected the filter algorithm should not be used, the estimate should instead the value indicated by the fuel level sensor(s) until the refill is done (parking brake released). When the refill is ended the algorithm continues to calculate using the current value from fuel level sensor(s) signal as initial value.
AER_201_20	When fuelRate input signal gets incorrect status ("Error" or "Not available") the filter algorithm should continue to calculate fuelLevelTot by only using fuelLevel as input K should be changed to $5 \cdot 10^{-5}$
AER_201_21	When fuelLevelSensor has status Error or Not Available the output should be the same. The state of Error or Not Available shall not be filtered. When the signal goes from Error or Not Available to a valid sample the filter shall be initialized with the value from the first correct sample.

Figure 3.13: Allocation Element Requirements for AE201

<b>ID</b>	<b>Description</b>
<b>AER_202_2</b>	The lowLevelFuelWarning should be set to 1(true) when input totalFuelLevel is below a predefined level. The level should be 10% for tank sizes equal or below 900 liters and 7% for tank sizes larger than 900 liters.
<b>AER_202_23</b>	The lowLevelFuelWarning should be kept true, once it is activated, until the algorithm is restarted by an ECU shutdown or if the totalFuelLevel reaches above 20%.
<b>AER_202_25</b>	Output signal lowLevelFuelWarning should have initial value 0 (false).

Figure 3.14: Allocation Element Requirements for AE202

# Chapter 4

## Compliance with ISO 26262

### 4.1 Approach

In Chapter 3 of the thesis, we developed and discussed the conceptual models corresponding to three of the most important work products from the Concept Phase of ISO 26262. These conceptual models give an essence of the requirements listed in the standard for their respective work products and give an overview of the requirements that a product needs to fulfill during its lifecycle to be in compliance with ISO 26262. To check if a product fulfills the requirements of the standard, we need to relate the functional and non-functional characteristics of the product with the requirements of the standard. For checking this, we use the conceptual models developed for the work products as reference models and verify the characteristics of the product under consideration by creating object diagrams of these work products for the product. The verification of whether the characteristics of the product fulfill the requirements of the standard is done automatically by the USE toolset. If any, conformance errors help us identify the loopholes in the design of our product according to ISO 26262. To

demonstrate this idea, we use FLEDS defined in Chapter 3 as a sample product.

We instantiate the work products by creating object diagrams using FLEDS as an example. Knowing the structure and functionality of FLEDS we create an object diagram for the class diagram of work product Item Definition. We then assume an example scenario that would lead to a hazardous event and create an object diagram for the work product Hazard Analysis and Risk Assessment. Knowing the hazard, we define a safety goal and create an object diagram for the work product Safety Goals. Then we check the object diagrams against the class diagrams of their respective work products. If the object diagrams of the work products are in conformance with the class diagrams, we say that the structure, functionality, hazardous events and safety goals for the hazardous events of FLEDS are defined according to the design requirements of ISO 26262 and FLEDS is in compliance with ISO 26262. However, a disagreement between an instance and its class diagram helps us find out where our design conflicts with the requirements of the standard and gives us an opportunity to design a product better compliant with the standard.

## 4.2 Example Scenario

To portray the idea of using class diagrams and object diagrams to test compliance of a product with the standard, we consider an example scenario of FLEDS.

Consider a truck running on liquid fuel on a highway with FLEDS implemented in it to keep a check on the fuel levels. Suppose that an error in the code for fuel level estimation causes an over estimation of fuel in the tank. We know that when the needle on the fuel gauge moves below a certain point or the bulb next to the fuel gauge lights up, the fuel tank needs to be refueled to keep the engine running. As a result of fuel level over estimation, the car runs out of fuel without giving a prior warning. Running out of fuel can damage the fuel pump and fuel filter of the car. In addition

to damaging the parts of a car, another important consequence of a vehicle running out of gasoline is the stalling of the vehicle. Let us assume that in our scenario the car stops moving, and not in a safe or controlled way. The driver loses the ability to accelerate or control speed. Due to the sudden stalling of the vehicle, the car running behind it in a high speed does not get enough time to stop and our vehicle comes into a collision with the car behind it. The sudden impact of a rear-end collision at such a high rate of speed causes devastating injuries. The collision causes serious injuries to the driver and the driver gets a herniated disc which could eventually cause paralysis to the driver. To avoid this situation, there is a need to ensure that the fuel levels in the vehicle are accurately measured and the fuel gauge shows the correct fuel level.

Now, to illustrate this scenario in the light of ISO 26262 we make use of the conceptual models of the work products mentioned in Chapter 3 of the thesis.

### 4.3 Item Definition

We have already defined a class diagram for Item definition in Chapter 3 of the thesis. Considering the example scenario mentioned above and taking into account the structure of FLEDS described in Chapter 3, we classify the structural components into the classes of Item Definition. The challenge that we face here is the identification of the parts of a FLEDS as the classes of Item Definition. For this purpose, we look back at the definitions in part 1 of ISO 26262.

Since the object under discussion is FLEDS, which is a standalone entity comprising of systems, we refer to it as an Item and all its subsystems as systems of this item. This classifies COO, EMS and ICL mentioned in Chapter 3 as systems of the item FLEDS. Another challenge here is defining the components of these systems as elements or E/E systems. For example, we know that the fuel level sensor is part of the COO system

and can be called an element of COO. But, at the same time, from the definition of E/E systems in the standard, we see that it can also be classified as an E/E system which is a

*“system (1.129) that consists of electrical and/or electronic elements (1.32), including programmable electronic elements.*

*EXAMPLE Power supply; sensor or other input device; communication path; actuator or other output device.”*

To clarify this confusion, we also refer to the definition of system, which states two notes:

*“NOTE 1 The related sensor or actuator can be included in the system, or can be external to the system.*

*NOTE 2 An element of a system can also be another system.”*

Note 2 here helps us in the classification because, although the fuel level sensor is an element of the COO system, it can also be another system and hence we classify it as an E/E system, further generalizing it as a sensor. Similarly, instead of classifying the allocation element AE201 as an element of the COO system, we call it a software component since a component itself is a non-system level element and can be tested outside of the system. Through this classification, we achieve an object diagram as shown in Figure 4.3. The detailed description of the abbreviations used in the object diagram is given in Appendix A. Note that the common attributes like serial number, value, etc., are not mentioned in the conceptual models as well as the object diagrams for the sake of brevity. The object diagram is created as an instance of the class diagram of Item Definition using USE specification. To check if this instance of Item Definition is in compliance with the conceptual model presented earlier, we use the model checking utilities of USE.

USE provides a detailed description of the structural inconsistencies between the object diagram and the class diagram. The error log for the instance of Item Definition is given in Figure 4.1.

```

Log
compiling specification C:\zahrasbackup\use-4.1.1\use-4.1.1\examples\Documentation\ISO26262-5\ItemDefinition.use...
done.
Model ItemDefinition (36 classes, 15 associations, 1 invariant, 0 operations, 0 pre-/postconditions, 0 state machines)
checking structure...
Multiplicity constraint violation in association `comprisesOf`:
  Object `AE202` of class `SoftwareComponent` is connected to 0 objects of class `SoftwareUnit`
  at association end `softwareUnit` but the multiplicity is specified as `1..*`.
Multiplicity constraint violation in association `contains`:
  Object `PBS` of class `Actuator` is connected to 0 objects of class `Element`
  at association end `element` but the multiplicity is specified as `1..*`.
Multiplicity constraint violation in association `contains`:
  Object `CANyellow` of class `Controller` is connected to 0 objects of class `Element`
  at association end `element` but the multiplicity is specified as `1..*`.
Multiplicity constraint violation in association `contains`:
  Object `Battery` of class `EESystem` is connected to 0 objects of class `Element`
  at association end `element` but the multiplicity is specified as `1..*`.
Multiplicity constraint violation in association `contains`:
  Object `EMS` of class `System` is connected to 0 objects of class `Element`
  at association end `element` but the multiplicity is specified as `1..*`.
Multiplicity constraint violation in association `contains`:
  Object `CANred` of class `Controller` is connected to 0 objects of class `Element`
  at association end `element` but the multiplicity is specified as `1..*`.
Multiplicity constraint violation in association `contains`:
  Object `CANGreen` of class `Controller` is connected to 0 objects of class `Element`
  at association end `element` but the multiplicity is specified as `1..*`.
Multiplicity constraint violation in association `contains`:
  Object `FuelLevelSensor` of class `Sensor` is connected to 0 objects of class `Element`
  at association end `element` but the multiplicity is specified as `1..*`.
checked structure in 0ms.
checking structure, found errors.
Ready.

```

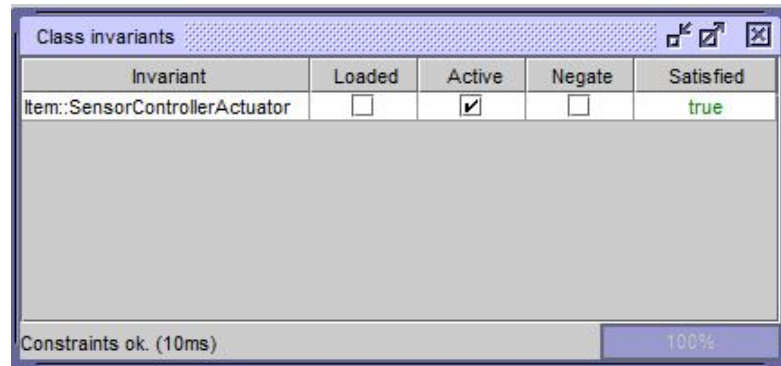
Figure 4.1: Error log for Item Definition

The error log from Figure 4.1 shows many errors specifically relating to the class *system* as we have not defined the components of the very basic systems such as the battery, sensor or the CAN cables. These errors can be ignored as it may not be important to mention the simplest of hardware components such as circuit boards and wires. However, in case of EMS, the log was useful in notifying that the components



of EMS have been ignored in the definition. Also, the software code for the allocation element AE202 has not been mentioned.

Upon checking if the OCL constraints that we defined for Item Definition were satisfied, we get the results as shown in Figure 4.2.



The screenshot shows a window titled "Class invariants" with a table of constraint verification results. The table has five columns: "Invariant", "Loaded", "Active", "Negate", and "Satisfied". The row for "Item::SensorControllerActuator" shows "Loaded" as an unchecked checkbox, "Active" as a checked checkbox, "Negate" as an unchecked checkbox, and "Satisfied" as the word "true" in green text. Below the table, it says "Constraints ok. (10ms)" and "100%".

Invariant	Loaded	Active	Negate	Satisfied
Item::SensorControllerActuator	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	true

Figure 4.2: OCL constraint verification for Item Definition

We see that our instance of Item Definition satisfies this constraint which ensures that FLEDS has a sensor, controller and an actuator in its system. Overall the verification of this instance of FLEDS against its conceptual model helps us identify that our instance of Item Definition has some missing elements and hence does not completely satisfy its model.

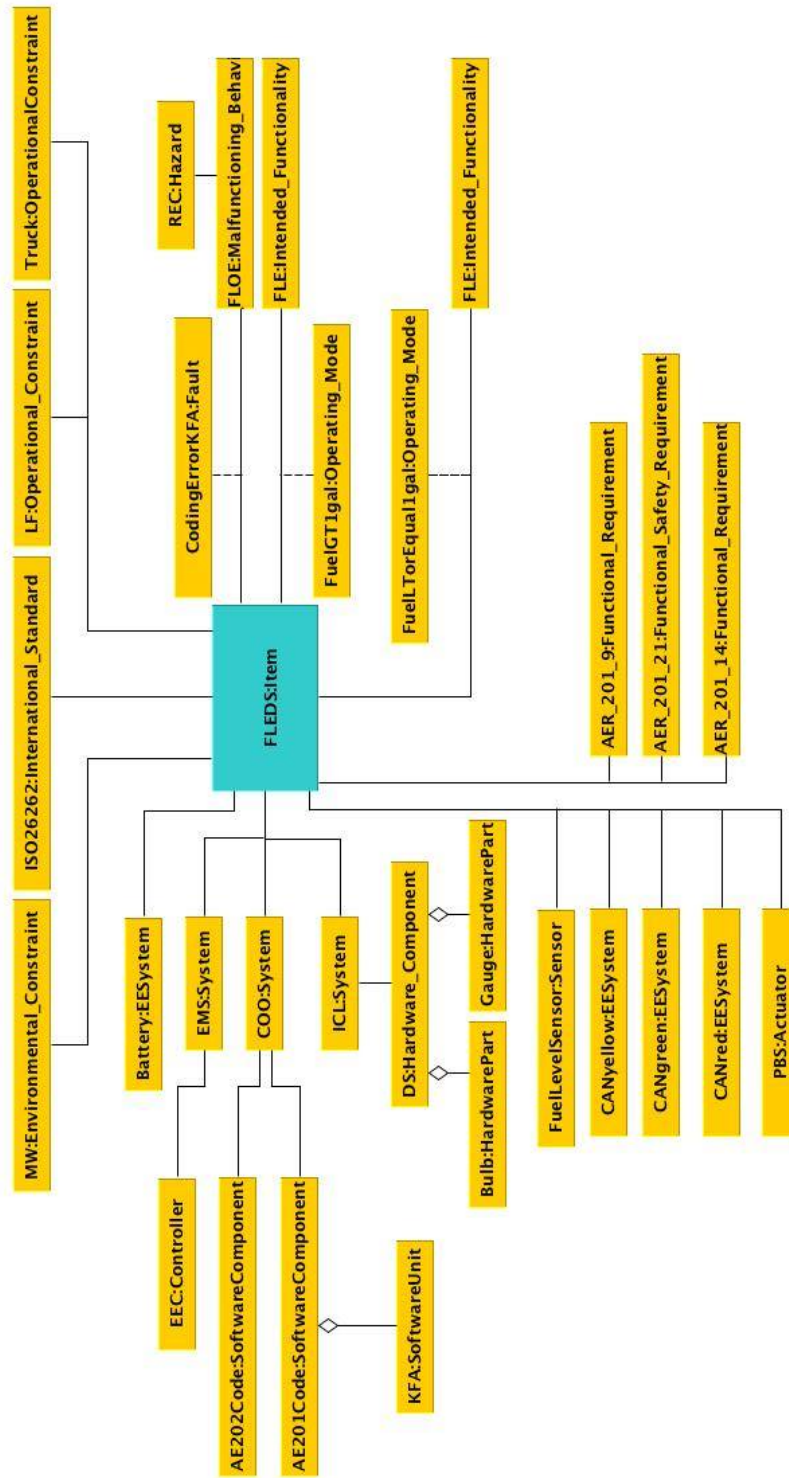


Figure 4.3: Object Diagram for Item Definition

## 4.4 Hazard Analysis and Risk Assessment

For the example scenario defined above for FLEDS, we carry out a hazard analysis and risk assessment. Suppose that through debugging of FLEDS we found out that there exists an error in the code of the Kalman Filter Algorithm (CodingErrorKFA:Fault) causing a miscalculation of fuel volume in the tank. This error caused a malfunction in the regular fuel estimation behavior causing a fuel level over estimation (FLOE:Malfunctioning\_Behavior) and displaying the fuel level higher than the actual fuel volume in the tank. As the fast moving vehicle suddenly runs out of fuel, the driver applies parking brakes (PBA:Emergency\_Operation) as an emergency operation that bring the vehicle to a safer condition (ReducedSpeed:Safe\_State). The malfunctioning behavior eventually results in the hazard of a rear end collision (REC:Hazard); the operational situation of the vehicle i.e., being driven on the highway (DHWY:Operational\_Situation) and the hazard combined, cause the hazardous event (RECDHWY:Hazardous\_Event). The driver suffered a whiplash (WPL:Harm) exposing him to the risk of being paralyzed (Paralysis:Risk). This hazardous event caused a disc herniation (NDH:Consequence) to the driver.

As this event resulted in the severe injury of the driver, we assign a severity class S3. Since the event was not controllable by the driver and the vehicle went out of control, the controllability class for this case is C3. As the probability of running below the normal fuel level is very high, therefore the exposure to this event is high leading us to assign a probability class E4. From Figure 3.8, this combination of attributes tells us that the hazardous event should have an ASIL of D. We assign these values to an instance of the conceptual model of HARA and get an object diagram as shown in Figure 4.7. The details about the values assigned to the objects in the object diagram can be found in Appendix A.

First we check the logs to see if the structure of our instance satisfies the structural aspects of the conceptual model of HARA. We see from Figure 4.4 that our instance is structurally valid.

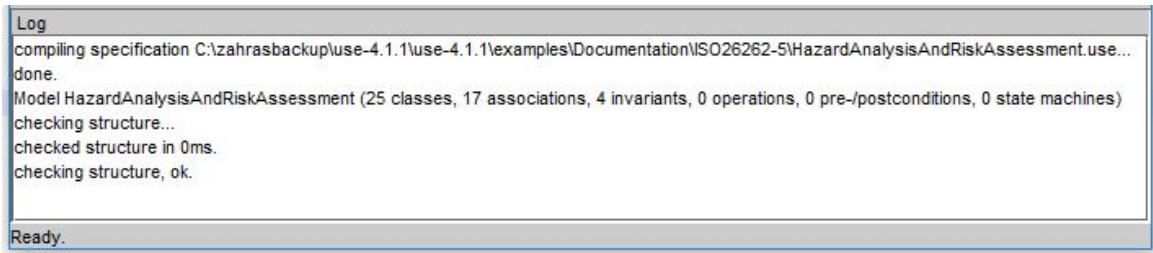


Figure 4.4: Error log for Hazard Analysis and Risk Assessment

To check if the correct ASIL has been assigned for this scenario, we checked the OCL constraints of the USE specification and got the results shown in Figure 4.5

Invariant	Loaded	Active	Negate	Satisfied
HazardousEvent::ASILDetermination	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	true
HazardousEvent::InScopeHazard	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	true
HazardousEvent::noASILifC0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	true
HazardousEvent::noASILifE0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	true
HazardousEvent::noASILifS0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	true

Constraints ok. (16ms) 100%

Figure 4.5: OCL constraint verification for Hazard Analysis and Risk Assessment

As all the constraints for this instance are satisfied, we can verify that the instance of HARA is in compliance with its conceptual model and the right ASIL has been assigned, thus enabling us to define the safety goals for this hazardous event.

## 4.5 Safety Goals

The purpose of the functional safety concept is to define a safety goal and then derive the requirements necessary to achieve that goal. In the given scenario, we know that a hazardous event occurred because of an error in the code which led to a false reading about the fuel level on the fuel gauge. Our goal is to prevent this from happening so the safety goal (SG) in this case would be to provide correct information about fuel level to the driver while the vehicle is moving.

Based on this safety goal, we derive our functional objective. One possible functional objective (FO) to achieve this goal could be to display the fuel level in the ICL gauge lower than the actual fuel level in the tank. This way the driver would know before the tank actually runs out of fuel. This functional objective leads us to define functional safety requirements and technical safety requirements. The process of deriving safety requirements for the elements of an item can be supported by safety analyses such as FMEA, FTA, HAZOP etc. This process is further evaluated in the work product Functional Safety Concept and is outside of the scope of WP Safety Goals, so we assume some functional safety requirements to check compliance of our model with the meta-model.

One FSR in this case would be to make a change in the software component of AE201 so that it shows the fuel level lower than what it actually is in the tank. We then evaluate the OCL constraints as shown in Figure 4.6 and determine that since the ASIL for the safety goal is the same as the ASIL evaluated for our hazardous event in HARA, our OCL constraints are satisfied.

A diagrammatic representation of this scenario is shown in Figure 4.8.

Invariant	Loaded	Active	Negate	Satisfied
HazardousEvent::ASILthenSafetyGoal	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	true
HazardousEvent::SameASILValues	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	true

Constraints ok. (0ms) 100%

Figure 4.6: OCL constraint verification for Safety Goals

## 4.6 Compliance

From the object diagrams presented in this chapter and the constraints checked for them against their respective conceptual models, we determine that the structure of FLEDS lacks some definition. The item definition of FLEDS does not define all the elements required according to its conceptual model and hence does not completely satisfy its meta-model. The hazard analysis however yields satisfactory results by fulfilling all requirements and hence we can say it complies with its model. The same is the case with safety goals. But, because these three work products are correlated and, for an item to be in compliance with ISO 26262, it should comply with all the work products' definitions, we say that this definition of FLEDS does not comply with ISO 26262.

We see that using conceptual modeling has provided us with a structured approach to check compliance of a product with ISO 26262. Using USE as a tool automated the process of checking the object diagrams against their respective class diagrams which reduced the time for compliance checking. With the use of USE, we were able to prove or disprove compliance for each of the work product and were also able to statistically analyze the discrepancies between the object models and their class diagrams.

Codes for the USE specification and their instances are provided in Appendix A.

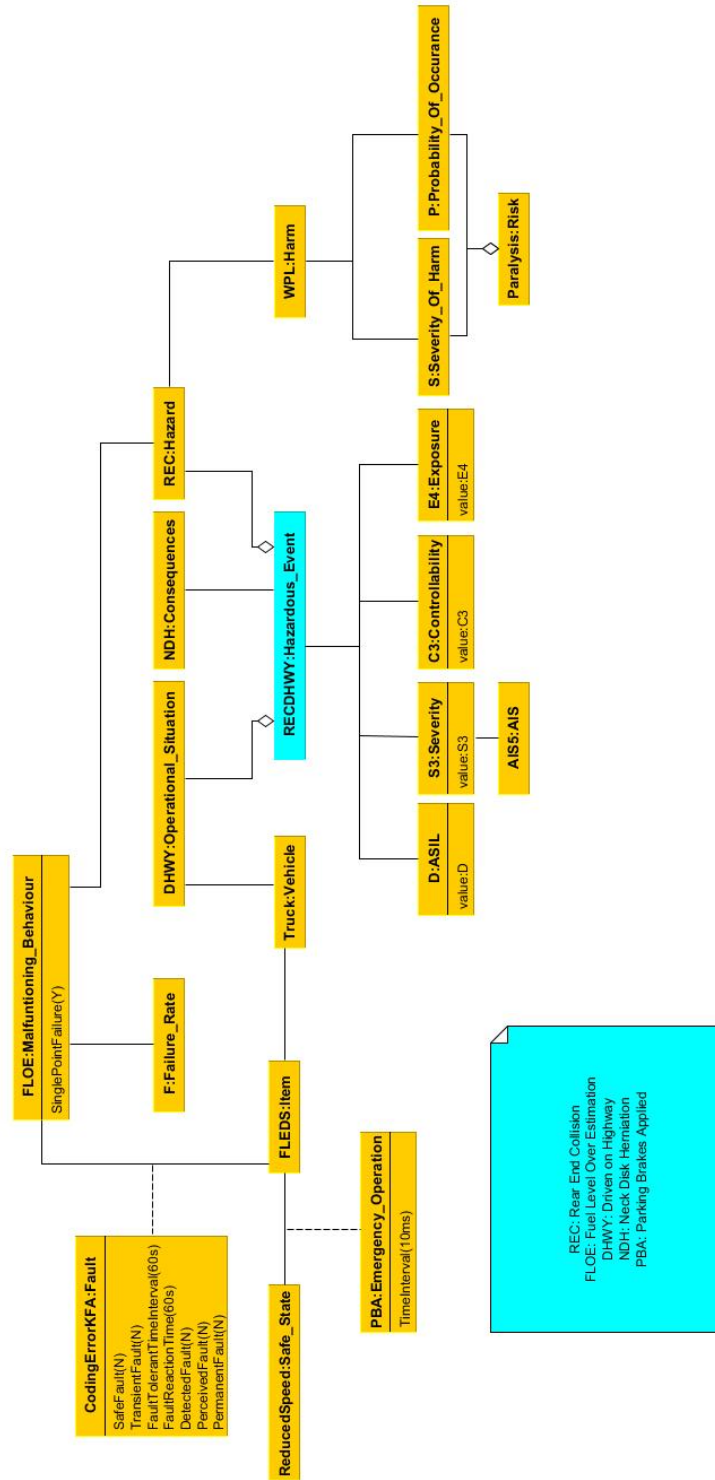


Figure 4.7: Object Diagram for Hazard Analysis and Risk Assessment



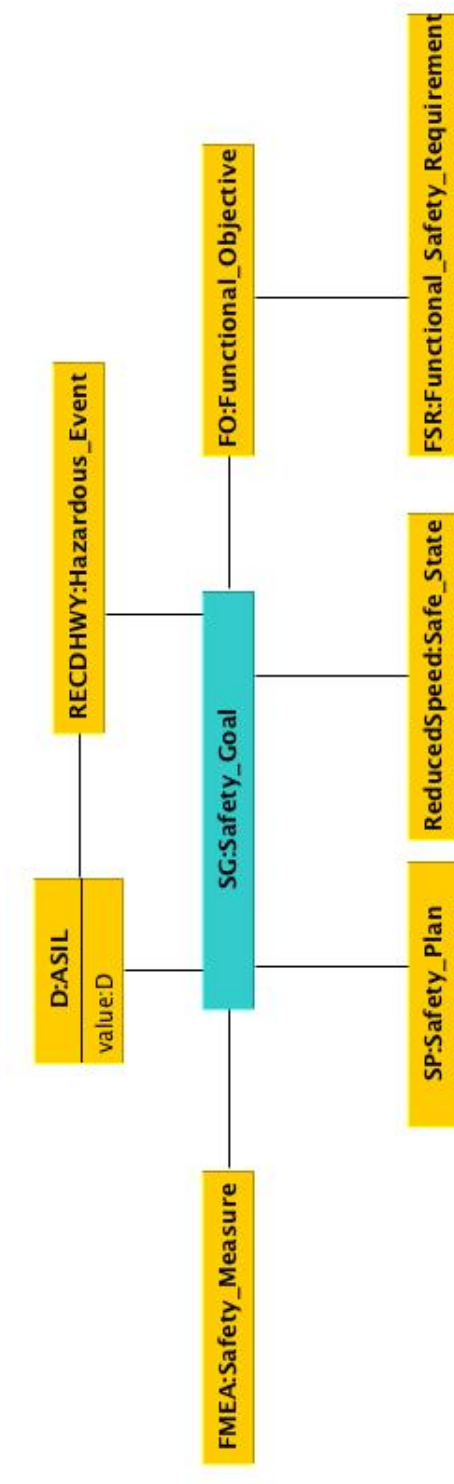


Figure 4.8: Object Diagram for Safety Goals

# Chapter 5

## Discussion

### 5.1 Challenges and Limitations

To design the conceptual models for work products, the first and most important step was to identify the concepts involved in a work product. This was a time consuming process as it required thorough understanding of the standard to extract vital information from the requirements and recommendations of a work product, classify it as a class, attribute or a relationship to present it in a formal notation. This was a cumbersome process as all of the concepts are not explicitly defined in the standard. Although each work product has a list of requirements and recommendations that need to be adhered to, a lot of the terms used in these requirements and recommendations are related to or derived from other concepts and some of the work products are prerequisites of other work products. It was therefore important to extract all the implicitly defined terms to completely represent a work product in UML notation. To demonstrate the use of conceptual modeling for product verification against ISO 26262, it was important to select the right work products that could illustrate the

complete idea. Since the standard has a total of 120 work products, it was a challenging task to pick the right work products to show the complete lifecycle of a product from design phase to determining its safety goals from an implementation point of view in the light of ISO 26262.

One important challenge while classifying the terms used in a work product was to identify the UML relationships between them. A lot of the terms were derived from other terms and concepts and hence there was a need to define the hierarchy and correlation of these terms. For this purpose, important design decisions had to be made and the standard had to be thoroughly understood for the terms to be strategically classified. We had to adopt a standard procedure to identify the relationships between the concepts of a work product as associations, aggregations, compositions and inheritance. Along with identifying the nature of the relationship between the terms, we also had to identify the restraints or regulations the standard defines for the terms. Some of the terms had relationship constraints while some constraints were to define the properties of these terms. The relational constraints were expressed through cardinality constraints, whereas the restraints to define the properties were expressed through the OCL. Since the constraints are vital to the structure of the standard, it was important to verify the relational and structural constraints so that the conceptual model rightfully replicates the guidelines mentioned for each work product.

Another challenge that we faced while classifying terms was that throughout the standard, a lot of terms had been used interchangeably. For example, the terms Hazard and Hazardous Event have been used with similar meanings at various places throughout the standard. Similarly, the term Fault has sometimes been referred to as Malfunctioning Behavior of an item or sometimes as Failure of an item. The usage of these terms in indistinguishable situations caused ambiguity about their definitions. To cope with this challenge, we referred to the vocabulary section of ISO 26262 which defines these terms and the relationships between them.

While defining object diagrams, we saw that the object diagrams lacked compliance with their conceptual models at some places, especially in the case of Item Definition, primarily because of not following the multiplicity constraints. This was due to the fact that we defined the multiplicity constraints as defined in the standard. For example, a E/E System being a type of System needs to one or more Elements defined. However, there are some situations where it is hard to categorize the E/E System at its atomic level, like in the case of a battery which classifies as an E/E System, it is impractical to list all its elements and components. Since it is not advisable to change the cardinalities to meet the product design requirements because the conceptual model would no longer be following the standard, it is the responsibility of the design and development team to either ignore the listings for lower level elements or to divide them in such sub classes so that it is no longer a requirement to mention all the parts of these elements.

Selecting the right tool and the right framework to demonstrate our concept was a big challenge, mainly because most tools lacked the support for OCL constraints. Although there is a wide variety of tools available for UML class diagrams, not all provide the framework to create object diagrams and verify their structure against their class diagrams. We also tried the Eclipse plugins for UML to demonstrate our idea, but even though some of them support OCL constraints, they lack the ability to apply the constraint at instance level and check the instance for conformance with its model.

## 5.2 Conclusion

In this thesis, we proposed an idea of using conceptual modeling to check compliance of a sample product from the automotive industry with ISO 26262. We presented the idea by generating conceptual models of some of the work products mentioned in the standard. The conceptual models were created using the requirements and guidelines

comprising the work products and were further supported by OCL constraints to incorporate the rules applicable in these work products. The concept was supported by instantiating the conceptual models using an example scenario. With the assistance of the right tools and technologies, we inferred that a product can be proved to be in compliance with ISO 26262 if it complies with the conceptual models set forth using the work products mentioned in the standard. This concept is not only an efficient way of compliance checking but also presents a clear view of the contents of the standard in a concise form.

The process of building conceptual models for work products of ISO 26262 helped us gain an insight into the standard and enabled us to critically analyze the contents of the standard. The models created not only gave us a picture of the requirements of the standard for product development and analysis but also helped us clarify and potentially remove the ambiguities that arose due to subjective interpretation of the standard. By expressing the concepts of ISO 26262 in terms of UML class diagrams, we were able to identify some instances that could cause uncertainty among the readers of the standard in terms of interpretation. At some points, the definitions presented in the first section of the standard for the terms used throughout the text of the standard were inconsistent with their implementation. This raised an increased need of a verified approach to systematize the conceptual models. The conceptual models developed in this thesis are plausible models of some of the work products of ISO 26262 but with the help of the expertise of people working closely with the design and development of ISO 26262 and its implementation in the industry, the models can be validated and agreed upon to eventually obtain models which are more accurate and reliable. This would not only provide us with a visual representation of the contents of ISO 26262 but will also provide us with a structured approach to validate our products against the industrial standard.

### 5.3 Future Work

ISO 26262 has about 120 work product definitions that need to be complied with for a E/E system to be deemed conformant with the standard. We have only represented our view of using conceptual modeling for compliance through three work products that could give the essence of our idea. Of course, in this aspect there is a lot of work that needs to be done. The conceptual models of the remaining work products spanning the whole standard need to be designed. A framework for the validation of these conceptual models needs to be created using analysis and design patterns to establish a standard interpretation. Since all the work products are related, their conceptual models need to be linked to each other by defining the right relationships and should be categorized by either being merged using model merging techniques or by being divided into UML packages. We also need to define model transformation strategies as the conceptual model goes through the different phases of the lifecycle of the E/E system and the properties that need to be preserved to ensure the conceptual models remain intact. Apart from the use of conceptual modeling, many other methods such as FMEA, STPA, FTA, HAZOP etc. need to be used for safety analysis to develop a set of functional safety requirements. Also, we need to define measurement scales according to which the values can be assigned to the quantifiable attributes. Though much needs to be done to present a complete picture of the standard, the idea of using conceptual modeling can certainly serve as a basis for checking compliance with ISO 26262.

# Appendix A

## Appendix

```
model ItemDefinition
```

```
--Classes
```

```
class Item  
end
```

```
class OperatingTime  
attributes  
TimeInMinutes : Integer  
end
```

```
class SimilarUnit  
end
```

```
class Constraint
end
```

```
class OperationalConstraint <Constraint
end
```

```
class EnvironmentalConstraint <Constraint
end
```

```
class MalfunctioningBehaviour
end
```

```
associationclass Fault
between
Item[1..*]
MalfunctioningBehaviour[1..*]
end
```

```
class Hazard
end
```

```
class Standard
end
```

```
class InternationalStandard <Standard
end
```

```
class NationalStandard <Standard
end
```



```
class IntendedFunctionality
end
```

```
associationclass OperatingMode
between
Item[1..*]
IntendedFunctionality[1..*]
end
```

```
class Assumption
end
```

```
class Requirement
end
```

```
class FunctionalRequirement <Requirement
end
```

```
class NonFunctionalRequirement <Requirement
end
```

```
class LegalRequirement <NonFunctionalRequirement
end
```

```
class Law <LegalRequirement
end
```

```
class Regulation <LegalRequirement
end
```

```
class FunctionalSafetyRequirement <FunctionalRequirement
end
```

```
class TechnicalSafetyRequirement <FunctionalSafetyRequirement
end
```

```
class System
end
```

```
class EESystem <System
end
```

```
class Sensor <EESystem
end
```

```
class Controller <EESystem
end
```

```
class Actuator <EESystem
end
```

```
class Element
end
```

```
class Component <Element
end
```

```
class SoftwareComponent <Component
```

end

```
class HardwareComponent <Component
end
```

```
class SoftwareUnit
end
```

```
class HardwarePart
end
```

```
class SafetyGoal
end
```

```
class Environment
attributes
FunctionalityRequired : String
end
```

```
--Associations association ItemOT between
Item[1..*]
OperatingTime[1..*]
end
```

```
association ItemSU between
Item[1..*]
SimilarUnit[0..*]
end
```

```
association ItemConstraint between  
Item[1..*]  
Constraint[1..*]  
end
```

```
association ItemStandard between  
Item[1..*]  
Standard[1..*]  
end
```

```
association ItemRequirement between  
Item[1..*]  
Requirement[1..*]  
end
```

```
association ItemSystem between  
Item[1..*] role relatedItem  
System[1..*] role relatedSystem  
end
```

```
association MBHazard between  
MalfunctioningBehaviour[1..*]  
Hazard[1..*]  
end
```

```
association FSRSafetyGoal between  
FunctionalSafetyRequirement[1..*]  
SafetyGoal[1..*]  
end
```

```
association Contains between
System[1..*]
Element[1..*]
end
```

```
association ItemEnv between
Item[1..*]
Environment[0..*]
end
```

```
--Aggregations aggregation ComprisesOf between
SoftwareComponent[1..*]
SoftwareUnit[1..*]
end
```

```
aggregation HComponentHPart between
HardwareComponent[1..*]
HardwarePart[2..*]
end
```

```
--Compositions composition IFAssumption between
IntendedFunctionality[1..*] Assumption[0..*]
end
```

```
--OCL constraints
constraints
```

```
--A system may relate a sensor, controller and an actuator with one another
```

context Item

inv SensorControllerActuator:

self.relatedSystem->exists(s |s.ocllsKindOf(Sensor)) and

self.relatedSystem->exists(s |s.ocllsKindOf(Controller)) and

self.relatedSystem->exists(s |s.ocllsKindOf(Actuator))

**-- Object Diagram : Item Definition**

```
!create FLEDS : Item
!create COO : System
!create EMS : System
!create ICL : System
!create Gauge : HardwarePart
!create Bulb : HardwarePart
!create CANyellow : Controller
!create CANred : Controller
!create CANGreen : Controller
!create KFA : SoftwareUnit
--Kalman Filter Algorithm
!create Battery : EESystem
!create PBS : Actuator
--Parking Brake System
!create AE201 : SoftwareComponent
!create AE202 : SoftwareComponent
!create FuelLevelSensor : Sensor
!create DS : HardwareComponent
--Display Screen
!create OT : OperatingTime
!set OT.TimeInMinutes := 60
!create MW : EnvironmentalConstraint
--Moderate Weather
!create Truck : OperationalConstraint
!create LF: OperationalConstraint
--Liquid Fuel
!create FLOE : MalfunctioningBehaviour
--Fuel Level Over Estimation
!create CodingErrorKFA : FailureMode between (FLEDS,FLOE)
```

```
!create REC : Hazard
--Rear End Collision
!create ISO26262 : InternationalStandard
!create FLE : IntendedFunctionality
--Fuel Level Estimation
!create FLW : IntendedFunctionality
--Fuel Level Warning
!create FuelGT1gal : OperatingMode between (FLEDS,FLE)
--Fuel level greater than 1 gallon
!create FuelLTOrEqal1gal : OperatingMode between (FLEDS,FLW)
--Fuel leevel less than or equal to 1 gallon
!create AER_201_9 : FunctionalRequirement
!create AER_201_11 : FunctionalRequirement
!create AER_201_12 : FunctionalRequirement
!create AER_201_21 : FunctionalSafetyRequirement
!create SG : SafetyGoal
```

```
!insert (FLEDS,OT) into ItemOT
!insert (FLEDS,ModerateWeather) into ItemConstraint
!insert (FLEDS,Truck) into ItemConstraint
!insert (FLEDS,Truck) into ItemConstraint
!insert (FLEDS,LiquidFuel) into ItemConstraint
!insert (FLEDS,ISO26262) into ItemStandard
!insert (FLEDS,AER_201_9) into ItemRequirement
!insert (FLEDS,AER_201_11) into ItemRequirement
!insert (FLEDS,AER_201_12) into ItemRequirement
!insert (FLEDS,AER_201_21) into ItemRequirement
!insert (FLEDS,COO) into ItemSystem
!insert (FLEDS,EMS) into ItemSystem
!insert (FLEDS,ICL) into ItemSystem
!insert (FLEDS,CANgreen) into ItemSystem
```



```
linsert (FLEDS,CANred) into ItemSystem
linsert (FLEDS,CANyellow) into ItemSystem
linsert (FLEDS,Battery) into ItemSystem
linsert (FLEDS,PBS) into ItemSystem
linsert (FLEDS,FuelLevelSensor) into ItemSystem
linsert (FLOE,REC) into MBHazard
linsert (AER_201_21,SG) into FSRSafetyGoal
linsert (COO,AE201) into Contains
linsert (COO,AE202) into Contains
linsert (AE201,KFA) into ComprisesOf
linsert (ICL,DisplayScreen) into Contains
linsert (DisplayScreen,Gauge) into HComponentHPart
linsert (DisplayScreen,Bulb) into HComponentHPart
```

**model** HazardAnalysisAndRiskAssessment

enum exposurelevel {E0, E1, E2, E3, E4}

enum asilevel {A, B, C, D}

enum controllabilitylevel {C0, C1, C2, C3}

enum severitylevel {S0, S1, S2, S3}

--Classes

class Item

end

class Vehicle

end

class OperationalSituation

end

class MalfunctioningBehaviour

attributes

SinglePointFailure : Boolean

end

class HazardousEvent

attributes

InScope : Boolean

end

```
class Hazard  
end
```

```
class Consequence  
end
```

```
class Risk  
end
```

```
class Harm  
end
```

```
class ProbabilityOfOccurrence  
end
```

```
class SeverityOfHarm  
end
```

```
class Severity  
attributes  
value : String  
end
```

```
class Controllability  
attributes  
value : String  
end
```

```
class Exposure
```

```
attributes
value : String
end
```

```
class asil
attributes
value : String
end
```

```
associationclass Fault
between
Item[1..*]
MalfunctioningBehaviour[0..*]
attributes
FaultTolerantTimeInterval : String
FaultReactionTime : String
SafeFault : Boolean
TransientFault : Boolean
SinglePointFault : Boolean
DetectedFault : Boolean
PerceivedFault : Boolean
PermanentFault : Boolean
end
```

```
class AIS
end
```

```
class SafeState
end
```

```
associationclass EmergencyOperation
between
Item[1..*]
SafeState[0..*]
attributes
TimeInterval : Integer
end
```

```
class RandomHardwareFailure <MalfunctioningBehaviour
end
```

```
class DependentFailure <MalfunctioningBehaviour
end
```

```
class CascadingFailure <DependentFailure
end
```

```
class CommonCauseFailure <DependentFailure
end
```

```
class FailureRate
end
```

```
--Associations
```

```
association BelongsTo between
Item[1..*]
Vehicle[1]
end
```

```
association Causes between
MalfunctioningBehaviour[1..*]
Hazard[1..*]
end
```

```
association HazardConsequences between
HazardousEvent[1..*]
Consequence[1..*]
end
```

```
association Sev between
HazardousEvent[1..*]
Severity[1]
end
```

```
association Exp between
HazardousEvent[1..*]
Exposure[1]
end
```

```
association Contr between
HazardousEvent[1..*]
Controllability[1]
end
```

```
association asilhazard between
HazardousEvent[1..*]
asil[0..1]
end
```

```
association ResultsIn between
Hazard[1..*]
Harm[1..*]
end
```

```
association ProbOfHarm between
Harm[1..*]
ProbabilityOfOccurence[1]
end
```

```
association SevOfHarm between
Harm[1..*]
SeverityOfHarm[1]
end
```

```
association Injury between
Severity[1..*]
AIS[1]
end
```

```
association Rate between
MalfunctioningBehaviour[1..*]
FailureRate[1..*]
end
```

```
--Aggregations
aggregation Includes between
HazardousEvent[1..*]
Hazard[1..*]
end
```

```
aggregation HEventOSituation between
HazardousEvent[1..*]
OperationalSituation[1..*]
end
```

```
aggregation CombinedInto between
Risk[1]
ProbabilityOfOccurence[1]
end
```

```
aggregation CombinedIntoRisk between
Risk[1]
SeverityOfHarm[1]
end
```

```
--OCL constraints
constraints
```

```
context HazardousEvent
--If hazardous event has E0 no asil is required
inv noasilifE0:
if self.exposure.value = 'E0' then self.asil->size = 0 else
self.asil->size = 1
endif
```

```
--If hazardous event has C0 no asil is required
inv noasilifC0:
if self.exposure.value = 'C0' then self.asil->size = 0 else
self.asil->size = 1
endif
```



--If hazardous event has S0 no asil is required

inv noasilifS0:

if self.exposure.value = 'S0' then self.asil->size = 0 else

self.asil->size = 1

endif

--asil determination

inv asilDetermination:

(self.severity.value = 'S1' and self.exposure.value = 'E3' and self.controllability.value = 'C3' implies self.asil.value = 'A') and

(self.severity.value = 'S1' and self.exposure.value = 'E4' and self.controllability.value = 'C2' implies self.asil.value = 'A') and

(self.severity.value = 'S2' and self.exposure.value = 'E2' and self.controllability.value = 'C3' implies self.asil.value = 'A') and

(self.severity.value = 'S2' and self.exposure.value = 'E3' and self.controllability.value = 'C2' implies self.asil.value = 'A') and

(self.severity.value = 'S1' and self.exposure.value = 'E4' and self.controllability.value = 'C3' implies self.asil.value = 'B') and

(self.severity.value = 'S2' and self.exposure.value = 'E3' and self.controllability.value = 'C3' implies self.asil.value = 'B') and

(self.severity.value = 'S2' and self.exposure.value = 'E4' and self.controllability.value = 'C2' implies self.asil.value = 'B') and

(self.severity.value = 'S3' and self.exposure.value = 'E3' and self.controllability.value = 'C3' implies self.asil.value = 'C') and

(self.severity.value = 'S3' and self.exposure.value = 'E4' and self.controllability.value = 'C2' implies self.asil.value = 'C') and

(self.severity.value = 'S3' and self.exposure.value = 'E4' and self.controllability.value = 'C3' implies self.asil.value = 'D') and

(self.severity.value = 'S1' and self.exposure.value = 'E1' and self.controllability.value = 'C1' implies self.asil.value = 'QM') and

(self.severity.value = 'S1' and self.exposure.value = 'E1' and self.controllability.value = 'C2' implies self.asil.value = 'QM')

```
context HazardousEvent
--if not in scope hazard classification is not necessary
inv InScopeHazard:
if self.InScope = true then self.asil->size = 1 else
self.asil->size >=0
endif
```

**-- Object Diagram : Hazard Analysis and Risk Assessment**

```
!create FLEDS : Item
!create Truck : Vehicle
!create FLOE : MalfunctioningBehaviour
--Fuel Level Over Estimation
!create CodingErrorKFA : Fault between (FLEDS,FLOE)
!set CodingErrorKFA.FaultTolerantTimeInterval := '60s'
!set CodingErrorKFA.FaultReactionTime := '60s'
!set CodingErrorKFA.SafeFault := false
!set CodingErrorKFA.TransientFault := false
!set CodingErrorKFA.SinglePointFault := false
!set CodingErrorKFA.DetectedFault := false
!set CodingErrorKFA.PerceivedFault := false
!set CodingErrorKFA.PermanentFault := false
!create DHWY : OperationalSituation
--Driven on Highway
!create REC : Hazard
--Rear End Collision
!create WPL : Harm
--Whiplash
!create P : ProbabilityOfOccurence
--assuming probability p = 0.5
!create S : SeverityOfHarm
--assuming severity S = 80 percent
!create Paralysis : Risk
!create RECDHWY : HazardousEvent
--Rear End Collision while being Driven on Highway
!create NDH : Consequence
--Neck Disk Herniation
!create E4 : Exposure
```

```
!set E4.value := 'E4'
!create C3 : Controllability
!set C3.value := 'C3'
!create S3 : Severity
!set S3.value := 'S3'
!create D : asil
!set D.value := 'D'
!create AIS5 : AIS
!create F : FailureRate
--assume F = 0.5

!insert (FLEDS,Truck) into BelongsTo
!insert (FLOE,REC) into Causes
!insert (RECDHWY,NDH) into HazardConsequences
!insert (RECDHWY,S3) into Sev
!insert (RECDHWY,E4) into Exp
!insert (RECDHWY,C3) into Contr
!insert (RECDHWY,D) into asil
!insert (REC,WPL) into ResultsIn
!insert (WPL,P) into ProbOfHarm
!insert (WPL,S) into SevOfHarm
!insert (S3,AIS5) into Injury
!insert (RECDHWY,REC) into Includes
!insert (RECDHWY,DHWY) into HEventOSituation
!insert (Paralysis,P) into CombinedInto
!insert (Paralysis,S) into CombinedIntoRisk
!insert (FLOE,F) into Rate
```

**model** SafetyGoals

--Classes

class Item  
end

class HazardousEvent  
end

class SafetyGoal  
end

class SafetyMeasure  
end

class SafetyMechanism  
end

class SafeState  
end

class FunctionalObjective  
end

class SafetyPlan  
end

```
class FunctionalSafetyRequirement
end
```

```
class asil
attributes
value : String
end
```

```
--Associations association HESafetyGoal between
HazardousEvent[1..*]
SafetyGoal[1]
end
```

```
association SGoalSMeasure between
SafetyGoal[1..*]
SafetyMeasure[1..*]
end
```

```
association SGoalSState between
SafetyGoal[1..*]
SafeState[1..*]
end
```

```
association SGoalFunctionalObjective between
SafetyGoal[1..*]
FunctionalObjective[1..*]
end
```

```
association SGoalSPlan between
SafetyGoal[1..*]
```

SafetyPlan[1]

end

association FObjectiveFSR between

FunctionalObjective[1..\*]

FunctionalSafetyRequirement[1..\*]

end

association SGoalasil between

SafetyGoal[1..\*]

asil[0..1]

end

association HEventasil between

HazardousEvent[1..\*] role hevent

asil[0..1] role asilLevel

end

--Aggregations aggregation SMeasureSMechanism between

SafetyMeasure[1..\*]

SafetyMechanism[1..\*]

end

--OCL constraints

constraints

context HazardousEvent

--asil of HazardousEvent shall be assigned to corresponding Safety Goal

inv SameasilValues:

self.asilLevel->notEmpty and self.sg.asil->notEmpty implies

self.asilLevel.value= self.sg.asil.value

--If Hazardous Event has an asil then it should have a safety goal

inv asilthenSafetyGoal:

self.asilLevel->notEmpty implies self.sg->notEmpty



**-- Object Diagram : Safety Goals**

```
!create SG : SafetyGoal
!create D : asil
!set D.value := 'D'
!create FMEA : SafetyMeasure
--Failure Mode and Effects Analysis
!create RECDHWY : HazardousEvent
!create FO : FunctionalObjective
!create FSR : FunctionalSafetyRequirement
!create ReducedSpeed : SafeState
!create SP : SafetyPlan
--assuming the company has a safety plan designed for hazards
```

```
!insert (RECDHWY,SG) into HESafetyGoal
!insert (SG,FMEA) into SGoalSMeasure
!insert (SG,ReducedSpeed) into SGoalSState
!insert (SG,FO) into SGoalFunctionalObjective
!insert (SG,SP) into SGoalSPlan
!insert (FO,FSR) into FObjectiveFSR
!insert (SG,D) into SGoalasil
!insert (RECDHWY,D) into HEventasil
```

# List of Acronyms

**AE** Allocation Element

**AER** Allocation Element Requirement

**AIS** Abbreviated Injury Scale

**ASIL** Automotive Safety Integrity Level

**CAN** Controller Area Network

**COO** Coordinator

**ECU** Electronic Control Unit

**E/E system** Electrical and/or Electronic System

**EMS** Engine Management System

**FMEA** Failure Mode Effects Analysis

**FTA** Fault Tree Analysis

**HARA** Hazard Analysis and Risk Assessment

**HAZOP** Hazard And Operability

**HW** Hardware

**ICL** Instrument Cluster

**IEC** International Electrotechnical Commission

**ISO** International Organization for Standardization

**MDE** Model Driven Engineering

**OCL** Object Constraint Language

**OMG** Object Management Group

**STPA** Systems Theoretic Process Analysis

**SW** Software

**QM** Quality Management

**UFR** User Function Requirement

**UML** Unified Modeling Language

**USE** UML-based Specification Environment

# References

- 1949329, S. T. P. D. (n.d.). *Allocation element requirement aer fuel level estimation: Ae201*.
- 1949330, S. T. P. D. (n.d.). *Allocation element requirement aer low fuel level warning: Ae202*.
- Brambilla, M., Cabot, J., & Wimmer, M. (2012). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1), 1–182.
- Buttner,, M. G. F., & Richters., M. (2007). *Use: A uml-based specification environment for validating uml and ocl. sci. of comp. prog.*, 69(1-3):27-34.
- Commission, I. E., et al. (2000). Functional safety of electrical/electronic/programmable electronic safety related systems. *IEC 61508*.
- Dardar, R. (2014). *Building a safety case in compliance with iso 26262 for fuel levelestimation and display system*.
- Gogolla, M. (2011). Uml and ocl in conceptual modeling. In *Handbook of conceptual modeling* (pp. 85–122). Springer.
- ISO 26262. (2011). Road vehicles-functional safety. *International Standard ISO/FDIS, 26262*.
- ISO 26262-1. (2011). Part 1: Vocabulary. *International Standard ISO/FDIS, 26262*.
- ISO 26262-10. (2011). Part 10: Guideline on iso 26262. *International Standard ISO/FDIS, 26262*.
- ISO 26262-3. (2011). Part 3: Concept phase. *International Standard ISO/FDIS, 26262*.

- Li, X. (2007). *Using uml for conceptual modeling: towards an ontological core* (Vol. 68) (No. 11).
- Luo, Y., van den Brand, M., Engelen, L., Favaro, J., Klabbers, M., & Sartori, G. (2013). Extracting models from iso 26262 for reusable safety assurance. In *International conference on software reuse* (pp. 192–207).
- Luo, Y., van den Brand, M., Engelen, L., & Klabbers, M. (2014). From conceptual models to safety assurance. In *International conference on conceptual modeling* (pp. 195–208).
- n.d., Y. T. D. (n.d.). *Yed graph editor*. retrieved. <https://www.yworks.com/products/yed>.
- Omg. *object constraint language, 2014. v2.4.* (n.d.). <http://www.omg.org/spec/OCL/2.4>.
- Omg *unified modeling language<sup>TM</sup>, 2015. v2.5.* (n.d.). <http://www.omg.org/spec/UML/2.5/>.
- Panesar-Walawege, R. K., Sabetzadeh, M., & Briand, L. (2013). Supporting the verification of compliance to safety standards via model-driven engineering: Approach, tool-support and empirical validation. *Information and Software Technology*, 55(5), 836–864.
- Pastor, O., & Molina, J. C. (2007). *Model-driven architecture in practice: a software production environment based on conceptual modeling*. Springer Science & Business Media.
- University, D. S. G. B. (2007). *Use: A uml-based specification environment*.