Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Master of Science (120 credits) in Computer Science with specialization in Software Engineering

INVESTIGATION OF AN OSLC-DOMAIN TARGETING ISO 26262

Focus on the left side of the Software V-model

Julieth Patricia Castellanos Ardila
jca13001@student.mdh.se

Examiner: Kristina Lundqvist
Mälardalen University, Västerås, Sweden

Supervisor: Barbara Gallina
Mälardalen University, Västerås, Sweden

Company supervisor: Barbara Gallina,
Scania, SE-151 87 Södertälje

September 27, 2016

Abstract

Industries have adopted a standardized set of practices for developing their products. In the automotive domain, the provision of safety-compliant systems is guided by ISO 26262, a standard that specifies a set of requirements and recommendations for developing automotive safety-critical systems. For being in compliance with ISO 26262, the safety lifecycle proposed by the standard must be included in the development process of a vehicle. Besides, a safety case that shows that the system is acceptably safe has to be provided. The provision of a safety case implies the execution of a precise documentation process. This process makes sure that the work products are available and traceable. Further, the documentation management is defined in the standard as a mandatory activity and guidelines are proposed/imposed for its elaboration. It would be appropriate to point out that a well-documented safety lifecycle will provide the necessary inputs for the generation of an ISO 26262-compliant safety case. The OSLC (Open Services for Lifecycle Collaboration) standard and the maturing stack of semantic web technologies represent a promising integration platform for enabling semantic interoperability between the tools involved in the safety lifecycle. Tools for requirements, architecture, development management, among others, are expected to interact and shared data with the help of domains specifications created in OSLC.

This thesis proposes the creation of an OSLC tool-chain infrastructure for sharing safety-related information, where fragments of safety information can be generated. The steps carried out during the elaboration of this master thesis consist in the identification, representation, and shaping of the RDF resources needed for the creation of a safety case. The focus of the thesis is limited to a tiny portion of the ISO 26262 left-hand side of the V-model, more exactly part 6 clause 8 of the standard: Software unit design and implementation. Regardless of the use of a restricted portion of the standard during the execution of this thesis, the findings can be extended to other parts, and the conclusions can be generalize.

This master thesis is considered one of the first steps towards the provision of an OSLC-based and ISO 26262-compliant methodological approach for representing and shaping the work products resulting from the execution of the safety lifecycle, documentation required in the conformation of an ISO-compliant safety case.

Acknowledgements

I would first like to thank my supervisor at the company and the University, Barbara Gallina, for her continuous support during the thesis research, her patience, motivation, and knowledge. Her guidance has helped me in the consecution of the best results and my own personal development.

I am grateful for having the opportunity to perform my master thesis as an industrial project at Scania AB. Being in this company gave me the practical knowledge that I have not experimented before.

Special thanks to Mattias Nyberg, ESPRESSO project leader, for giving me the opportunity to participate and contribute to the project.

I am thankful with Kathyayani Padira, student/colleague at Scania. We participated in interesting discussions and worked together during the learning process. Her thesis outcomes and her experiences helped in the consecution of my thesis results.

Furthermore, I would like to acknowledge and mention the two projects within which this master thesis was conceived, ESPRESSO [1] and Gen&ReuseSafetyCases [2].

I also want to give special thanks to my mother Mercedes for always believe in me, offering her most caring support and enthusiasm.

Finally, and most important, I would like to express my gratitude and love to my husband Ola and my little son Gabriel. Their company, hugs, and unconditional support have strengthened me through this challenging experience.

Table of Contents

List of Figures

List of Tables

Acronyms

AE Allocation Element.

AM Architectural Management.

ASIL Automotive Safety Integrity Level.

COO Coordinator.

dcterms Dublin Core Metadata Initiative Terms.

E/E system electrical and/or electronic system.

ECU Electronic Control Unit.

FLDS Fuel level Display System.

FLEDS Fuel Level Estimation and Display System.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

IRI Internationalized Resource Identifier.

ISO International Standarization Organization.

LOV Link Open Vocabularies.

OSLC Open Services for Lifecycle Collaboration.

OWL Web Ontology Language.

QM Quality Management.

RDF Resource Description Framework.

RDFS Resource Description Language Schema.

ReSh Resource Shape.

RM Requirements Management.

SHACL Shapes Constraint Language.

ShEx Shape Expressions.

SPARQL Protocol and RDF Query Language.

SPIN SPARQL Inference language.

UML Unified Modeling Language.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

W3C World Wide Web Consortium.

WWW World Wide Web.

XML eXtensible Markup Language.

XSD XML Schema.

1 Introduction

This chapter represent the thesis introduction and is organized as follows: Section 1.1 presents the motivation for the thesis, section 1.2 presents the context in which this thesis is defined, section 1.3 presents the contributions of the thesis, and the section 1.4 presents the thesis outline.

1.1 Motivation

ISO 26262 [5] is a standard that addresses the functional safety of road vehicles. The current version (ISO 26262:2011) applies to cars with a maximum gross vehicle mass up to 3500 kg, which does not include the kind of vehicles produced at Scania. So, the compliance with the standard is currently not carried out at the company. However, it is likely that by 2018, Scania will adopt the regulations prescribed by the norm [13, 3], since a new version (ISO 26262-Edition 2) is, at present, extending the coverage to other kinds of vehicles e.g. motorcycles, trucks, and buses [16]. ISO 26262 *"defines a safety lifecycle to be adopted during the development of automotive safety-critical systems"* [17], and one of the requirements to be in compliance with the standard is the provision of a safety case. A safety case is *"a structured argument, which inter-relates evidence and claims, needed to show that safety-critical systems are acceptably safe"* [18]. Traceability, which is defined as *"the degree to which a relationship can be established between two or more products of the development process"* [19], becomes an essential property with the application of ISO 26262 since its role in providing evidence is crucial for the creation of a safety case. The safety case and the safety lifecycle are tightly coupled, as the evidence required to be collected in the safety case is produced during the safety lifecycle activities. A well-performed documentation process would lead to the creation of a consistent safety case, and thus, the safety assessment would not be at risk.

The adoption of ISO 26262 in a company like Scania would be exceedingly time-consuming for the personnel in charge of its application, *"where hundreds of documents are expected to be the result of the safety lifecycle activities"* [20]. Performing the documentation process manually would also be chaotic, because safety information will come from different sources and involve various stakeholders. For this reason, the use of software tools is considered necessary for supporting the documentation process. However, in most of the cases, these tools do not have standardized methods for sharing data. The semantic web technologies and specifically the Open Services for Lifecycle Collaboration (OSLC) standard represent a promising integration platform for enabling the capacity of tools for working together. OSLC is defined as *"an open community creating specifications for integrating tools"* [21], offering the opportunity to use domains already created or participate in the formulation of domain specifications for lifecycle interactions. Domains specifications formulated in OSLC will enhance the data exchange between lifecycle tools.

The aim of this thesis is to offer an OSLC-based infrastructure enabling the automatic generation of safety case fragments. More specifically, the purpose of this thesis is the identification, representation, and shaping of resources needed to create a safety case. The focus is limited to a tiny portion of the ISO 26262 left-hand side of the V-model: *the software unit design specification.* For reaching the purpose of this thesis, the following questions will be addressed:

1. What are the ISO 26262 requirements for the selected tiny portion of the standard required to be modeled as metadata elements?
2. How can an OSLC domain specification be reused or built to incorporate the metadata elements defined?
3. How can OSLC resources be shaped, so the data provided to the metadata elements is also correctly specified?

1.2 Context

This master thesis is carried out at Scania in Södertlje, Sweden. Scania [22] is an international automotive industry manufacturer of commercial vehicles, specialized in heavy trucks and buses. Scania, as a global company, has operations in Europe, Latin America, Asia, Africa and Australia, and its sales and service organizations are present in more than 100 countries.

Solutions for documentation management are currently being investigated at Scania. Part of these research initiatives are Gen&ReuseSafetyCases project [2] and the Vinnova ESPRESSO project [1]. ESPRESSO's aim is to contribute in converting Scania documentation into machine-readable formats, so documentation management is more efficient and more aligned to ISO 26262, while Gen&ReuseSafetyCases' purpose is the creation of the safety case based upon model-based safety certification ideas, in compliance with ISO 26262 [20]. This master thesis is conceived in the context of these two interrelated projects and, also, has benefited from discussions held in the context of the EU ECSEL AMASS (Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems) project [23].

1.3 Contribution

The present master thesis will contribute to enabling the semi-automation of the creation of a safety case, investigating how a portion of the safety lifecycle proposed by ISO 26262 can be semantically interconnected adopting the OSLC standard. The main outcome of this thesis is a report that includes:

1. An analysis of a portion of ISO 26262 safety lifecycle for defining the terms and relationships required to build metadata elements.
2. The identification of an approach for building OSLC domains.
3. A comparative study targeting three existing RDF constraint languages for validating RDF data.
4. A tiny but self-contained OSLC-domain targeting ISO 26262 part 6 and Scania needs, shaped according to the findings stemming from 3.

As a result of early work of this thesis, a scientific article [17] was presented and accepted at the workshop on Critical Automotive applications: *Robustness & Safety (CARS 2016)* in Gothenburg, Sweden.

1.4 Document outline

Chapter 2 presents the background and related work where principles and terminologies related to ISO 26262 and semantic web, among others, are explained in the level of detailed required for the understanding of the rest of the thesis.

Chapter 3 presents the scientific research method in which this master thesis is based. The methodology used is based on qualitative research approaches and the application of a case study.

Chapter 4 presents the problem formulation and analysis. In this chapter, the main problem is described in detail and then divided into smaller subproblems, where each of them is covering a certain aspect of the main problem.

Chapter 5 presents the methods used in this thesis to solve the problem. These methods are related to the definition of OSLC domain specification development, and the identification of constraint languages for shaping data

Chapter 6 presents the solution reached by addressing this thesis. This solution includes a metamodel, which includes the elements and relationships found in the clause 8 of ISO 26262:6 and the Scania practices, an OSLC 26262-compliant specification, and a SHACL schema for shaping the ISO 26262-compliant resources.

Chapter 7 presents the applicability of the solution, using the software unit safety case on Fuel Level Estimation Algorithm.

Chapter 8 presents the conclusions reached by performing this master thesis and gives indications of what future work can be done to improve the results.

2 Background and related work

This chapter introduces the background underpinning the current research, helping in the understanding of its content. The chapter is organized as follows: Section 2.1 presents an overview of the concepts and terms related to system safety, Section 2.2 presents an overview of the fuel level estimation system and the software unit fuel level estimation algorithm, Section 2.3 presents ISO 26262 related concepts, Section 2.4 presents documentation management, Section 2.5 presents traceability and interoperability, Section 2.6 presents semantic knowledge representation, Section 2.7 presents semantic web fundamentals, Section 2.8 presents RDF data shapes, Section 2.9 presents the main characteristics of OSLC, and Section 2.10 presents the related work.

2.1 System safety

This section presents the concepts related to safety systems. It is structured in the following way: Subsection 2.1.1 provides the basic terms required for the understanding of rest of the section, Subsection 2.1.2 presents an overview of safety-critical systems, and Subsection 2.1.3 presents an overview of functional safety.

2.1.1 Basic terms

This subsection recalls some terms required in the understanding of system safety. The terms are organized alphabetically.

Correct service is delivered when the service implements the system function [24].

Failure or service failure, is a transition from correct service to incorrect service (not implementation of the system function) [24].

Function is what the system is intended to do and it is described in the functional specification in terms of functionality and performance [24].

Hazard is an unsafe condition, which can cause harm [3].

Residual risk is the risk remaining after the deployment of safety measures [5].

Risk is the combination of the probability or occurrence of harm and the severity of that harm [5].

Safety is the absence of catastrophic consequences on the user(s) and the environment [24].

Safety-related is any equipment (with or without software) whose failure contribute to a hazard [25].

System is an entity that interact with other entities [24].

2.1.2 Safety critical systems

Safety and mission-critical systems, like those found in automotive, nuclear and chemical plants or aerospace domains, are systems that in principle should never fail [26], since a failure might result in the loss of human life, damage environment or property, or cause of severe injury [27]. In general, a system is safety-critical *"when we depend on it for our well-being"* [28]. In practice, hundred percentage of safety is an unreachable goal, so critical domains are subjected to meticulous assurance process to guarantee that the system is safe enough in a determined context [29]. Safe enough or acceptably safe is a condition that is reached when the residual risk is reduced to its maximum and the risk is acceptable. So, when acceptably safe is claimed, the system can operate safe in a specified environment [3].

Safety critical systems (and their specifications) are based on very complex software systems, and the determination of possible causes or consequences of failures is extremely difficult [30], even

though there is a landscape of software failure cause models that can be applied [31]. Safety critical systems carry out high-risk functions and for this, they normally need to be qualified, certified, and follow rigorous development methodologies to assure their integrity. These methodologies, according to [32], must ensure the understanding of the requirements, the precision of the specifications, and the existence of adequate metrics to validate and verify their quality.

2.1.3 Functional Safety

Functional safety [33] is the part of the overall safety of a system, where the system functions are addressed to ensure that they work correctly in response to their inputs. Functional safety focuses on electronic systems, their related software, and their related hardware. The goal of the functional safety is to minimize the risks associated to the system's functionalities, reducing them to a tolerable level and minimizing their negative impact [25]. A safety analysis, which includes a comprehensive and systematic examination of the system [34], is carried out to identify risks. Once the risks are identified, the functional safety process establishes a maximum tolerable frequency for each failure mode. This rate leads to the generation of safety integrity levels that are assigned to the safety-related items. Safety-related items must carried out corrective and/or preventive actions [25].

The functional safety assurance is guided by the application of safety standards. Safety standards specify, in a precise way, the kind of activities that must be followed for guarantying an acceptable level of safety. Safety critical automotive systems, which is the focus of this thesis, count on activities defined by the standard ISO 26262, to reach functional safety [3]. Details of ISO 26262 can be found in Section 2.3.2.

2.2 Fuel Level Display System (FLDS)

This section aims to provide information about the case study used in the context of this thesis, Fuel Level Display System (FLDS), with the focus on the specific parts of the system that are related to the thesis, *the fuel level estimation algorithm*. The information will be mainly taken from [4] and technical product data reports from Scania. This section is structured in the following way: Subsection 2.2.1 presents the basic terms required for understanding the rest of the section, Subsection 2.2.2 presents an overview of FLDS, and Subsection 2.2.3 presents the fuel level estimation algorithm.

2.2.1 Basic terms

This subsection presents a lists of terms required for the understanding of the overall section. The terms are organized alphabetically.

Kalman algorithm is a data fusion algorithm. Typical uses of the Kalman algorithm include smoothing noisy data and providing estimates of parameters of interest [35].

Simulink is a block diagram environment for multidomain simulation and Model-Based Design. It supports simulation, automatic code generation, and continuous test and verification of embedded systems[1].

2.2.2 FLDS overview

FLDS is one of the systems required for manufacturing the trucks and buses at Scania. The system's purpose is to distribute fuel level information so that the fuel level can be monitored [36]. Additionally, the system shows the fuel level to the driver and activates an alarm when the fuel level is low so that the driver can plan the refueling on time. FLDS is used in both buses and trucks functioning with liquid or gas fuel engine. There are four versions of the system (called variants) configured according to the needs of different kind of vehicles. FLDS-variant one is composed by four ECUs (See Figure 1).

[1] `http://se.mathworks.com/products/simulink/`

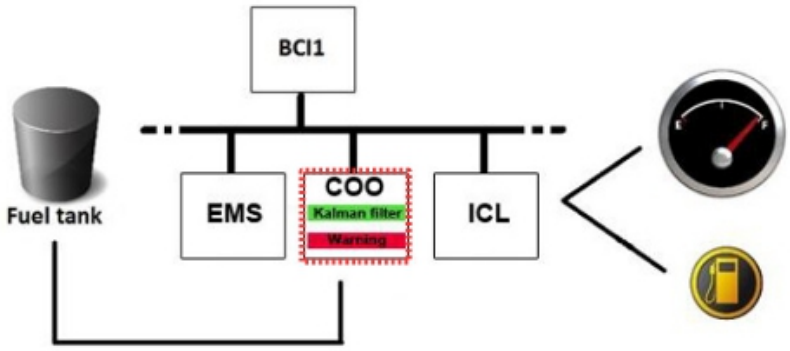


Figure 1: ECUs configuration for variant 1 ([3]).

COO, the ECU highlighted with a red-dotted line in Figure 1, is in charge of calculating the fuel level estimation using a Kalman algorithm. Its design and implementation are carried out as a model in Simulink. This model is considered *"as user defined blocks that store signals on both input and output sides* [4]. A representation of the COO model is depicted in Figure 2, where the software unit *fuel level estimation algorithm* (the software unit that is the focus of this thesis) is highlighted with a red-dotted line. The fuel level estimation algorithm is studied in subsection 2.2.3.

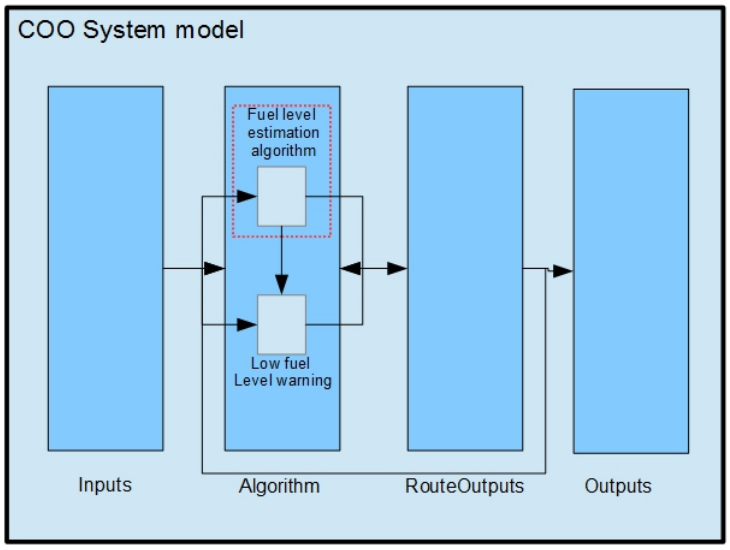


Figure 2: A representation of the COO system model (adapted from [4]).

2.2.3 Fuel level estimation algorithm

Fuel level estimation algorithm is a software unit in charge of the calculation of the fuel level. The entire block is defined by input ports, output ports, unit functions, subfunctions, input arguments and return values for the functions, configuration parameters, parameters values and constraints. For respecting the Scania's privacy policy, just the required elements to built a complete case will be shown in this section. First, the software unit is fed by three input port signals, namely *fuel_params_InputBus_str*, *fuel_InputBus_str* and *tank_Capactity_str*. The unit produces as a result three output signals *fuel_Volume_str*, *fuel_LevelTot_str* and *reset_str*. Inside the software unit, five software unit functions are present. One of the units is detailed, e.g., *CalculatCurrentVolumeLevels*. The input and output ports mentioned are presented in figure 3, and the software unit function considered is highlighted by a dotted-red line for being detailed later.

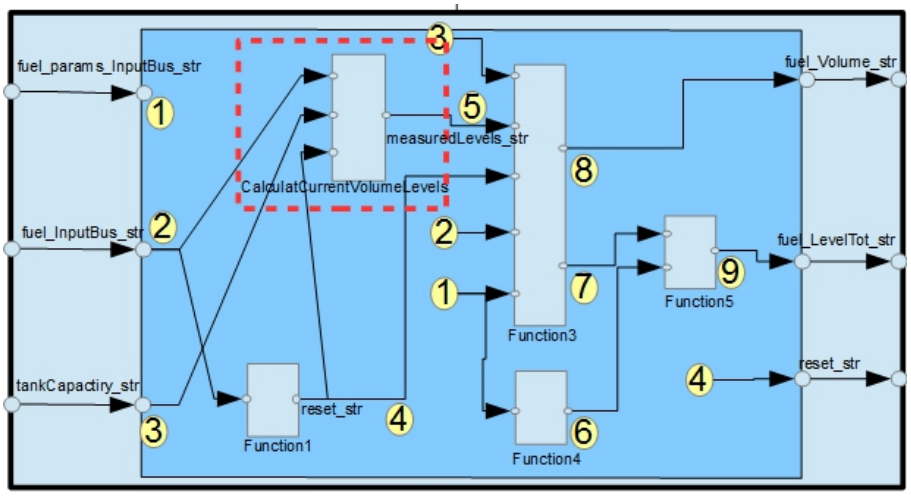


Figure 3: Details of the Software unit Fuel Level Estimation Algorithm.

The names of the variables finish in _str. This is a conventional way used at Scania for defining the data type of the variables. In this case, the ending _str means that the datatype of the variables is String. The software unit function selected for the modelling (and highlighted in Figure 3 with a red-dotted line) is called *CalculatCurrentVolumeLevels* (see Figure 4)

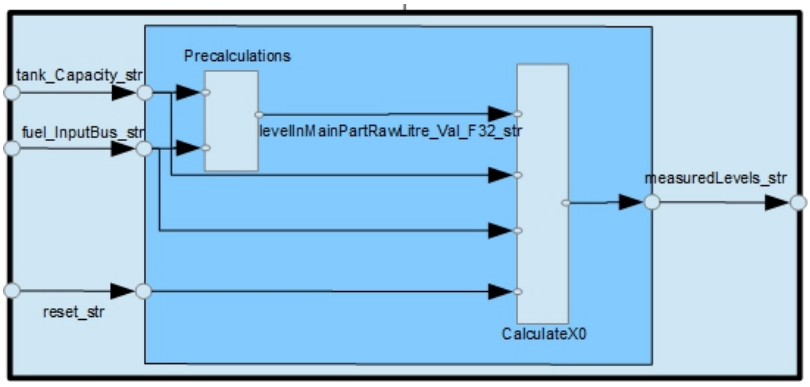


Figure 4: Software unit Function CalculatCurrentVolumeLevels.

CalculatCurrentVolumeLevels is a software unit function in charge of mapping the fuel level sensor depending on the tank variant, and defining the x0 state, which is the startup state of the Kalman filter. The unit receives three input arguments, namely *tankCapacity_str*, *fuel_inputBus_str* and *reset_str*. The first two input arguments are derived from the input ports of the software unit that have the same names. The last one is calculated inside the unit by other software unit function. The returned value of the function is called *measuredLevels_str*. This return values corresponds to the fuel current level calculated by the function. There are two subfunctions inside the function *CalculatCurrentVolumeLevels*, namely *Precalculations*, which is in charge of doing some calculations required for finding the measured levels, and *calculateX0* the subfunction in charge of the startup state for the Kalman filter algorithm. The first subfunction receive two input arguments, namely *tank_Capacity_str* and *fuel_inputBus_str*. The processing in the software unit subfunction has a result the return value called *levelInMainPartRawLitre_Val_F32_str*. The second subfunction has as input arguments *tank_Capacity_str*, *fuel_inputBus_str*, *reset_str* and *levelInMainPartRawLitre_Val_F32_str*, and the return value is *measuredLevel_str*.

Fuel Level estimation algorithm corresponds to the software unit that realizes the specifications presented in the allocation element document called *AE201* (an internal document at Scania). This document has more detailed information about the software unit, for example, configuration parameters are defined and the signals (which in the document are called variables) are accompanied

with information like constraints. For this variant, the detailed information of the signals shown in Figure 3 are the following:

- **fuel_params_inputBus_str**: This variable is required to storage the fuel parameters values retrieved from the Real-time Database.
- **fuel_inputBus_str**: Fuel values obtained from the fuel level sensors and stored in the Real-time Database.
- **tankCapacity_str**: Read the capacity of the fuel tank. Tank capacity has a required range from 0 to 2500, and a required accuracy of 1 and unit L (liters).
- **fuelVolume_str**: Estimated total fuel level converted to liters.
- **fuelLevelTot_str**: Estimated total fuel level in the tank. The fuelLevelTot_str has a required range from 0 to 100, required accuracy of 0.4 and unit is %.
- **reset_str**: It is a variable that provides information about the good status of the signals, so loose connector do no affect the calculations of the algorithm.

Two Configuration Parameters are also present in the document. They are called *fuelLevelSensorParam* and *fuelLevelTotalParam*. Both Configuration parameters have parameter values, but it is only presented here the parameter values for *fuelLevelTotalParam* (see Table 1).

Table 1: Configuration Parameters for variant 1.

Name	**Description**	**Possible values**	**Value description**
fuelLevelTotalParam	Describes the source of the value, controlling if the total fuel estimation is done in CMS or received via CAN	10	Total fuel level calculated by COO

2.3 ISO 26262

This section aims to provide information about the ISO 26262:2011-compliant lifecycle, with the focus on the parts of the standard that will be treated in this thesis. The information will mainly be taken from the standard ISO 26262 [5]. This section is structured in the following way: Subsection 2.3.1 presents a list of essential terms required for the understanding of the ISO 26262, Subsection 2.3.2 presents an overview of ISO 26262:2011, and Subsection 2.3.3 presents the software unit design and implementation, the part of the standard in which this thesis is interested.

2.3.1 Basic terms

This subsection recalls some terms required for the understanding of the standard ISO 26262 and the specific sections that are part of this thesis. The terms are provided by the standard ISO 26262-1: Vocabulary [37], and they are organized alphabetically.

ASIL which means Automotive Safety Integrity Level, is one of four levels to specify the items or elements necessary requirements of ISO 26262 and Safety measures to apply for avoiding an unreasonable residual risk, with D representing the most stringent and A the least stringent level.

ASIL decomposition is the apportioning of safety requirements redundantly to sufficiently independent elements, with the objective of reducing the ASIL of the redundant safety requirements that are allocated to the corresponding elements.

Item is a system or array of systems to implement a function at the vehicle level, to which ISO 26262 is applied.

Lifecycle is the entirety of phases from concept through decommissioning of an item.

Safety goal is the top-level safety requirement as a result of the hazard analysis and risk assessment.

Safety measure is an activity or technical solution to avoid or control systematic failures and to detect random hardware failures or control random hardware failures, or mitigate their harmful effects.

Software unit is the atomic level software component of the software architecture that can be subjected to stand-alone testing.

Unreasonable risk is the risk judged to be unacceptable in a certain context according to valid societal moral concepts.

Work product is the result of one or more associated requirements of ISO 26262.

2.3.2 ISO 26262:2011 overview

ISO 26262 [5] is a standard that addresses the functional safety of the electrical and/or electronic systems (E/E systems) included in road vehicles with a maximum gross mass up to 3500 kg. Additionally, ISO 26262 ensures the safety of safety-related elements based on other technologies e.g. the vehicle's software systems. ISO 26262 is an adaptation of the standard IEC 61508 (IEC stands for International Electrotechnical Commission) and provides a framework where an automotive safety lifecycle is defined. ISO 26262 safety lifecycle pays special attention to the automotive-specific risks and formulates precise procedures to avoid unreasonable risk. Procedures are applicable requirements of ISO 26262 defined according to the ASIL. ASIL values are determined during the hazard analysis (*"a process of recognizing hazards that may arise from a system or its environment, documenting their unwanted consequences and analyzing their potential causes"* [2]) and assigned to the safety goals of an item under development. If no ASIL decomposition is performed, ASIL values are propagated throughout the items lifecycle so, subsequent safety requirements, architectural elements, hardware and software inherit the ASIL. For claiming compliance with ISO 26262 each requirement provided by the standard shall be complied with unless tailoring (omitted or performed in a different manner) of the safety activities in accordance with ISO 26262 has been planned, or a rationale where an explanation of non-compliance is available. The rationale, if exists, has to be assessed in accordance with ISO 26262. ASIL values can change if ASIL tailoring is applied during the lifecycle. ASIL tailoring is called ASIL decomposition, and it is allowed under certain conditions. Recalling from the standard:

"If ASIL decomposition is applied at the software level, sufficient independence between the elements implementing the decomposed requirements shall be checked at the system level and appropriate measures shall be taken at the software level, or hardware level, or system level to achieve sufficient independence."

ISO 26262 comprises ten parts that provide an automotive safety assurance and supporting activities. The parts, shown in Figure 5, are the following: the vocabulary, part 1; the management of functional safety, part 2; the concept phase, part 3; the product development at the system level, part 4; the product development at the hardware level, part 5; the product development at the software level, part 6; the production and operation, part 7; the supporting activities, part 8, ASIL-oriented and safety-oriented analysis, part 9; and the guidelines on ISO 26262, part 10. The results of safety activities are known as work products. The work products that are required at the beginning of every stage of the lifecycle are known as prerequisites. The lifecycle is defined in ISO 26262 from the part 3 to the part 7 and the activities are distributed in a V-shape model, which is followed from the left side of the V to the right.

[2] `http://www.chambers.com.au/glossary/hazard_analysis.php`

The main portion of the standard ISO 26262 that will be studied in this thesis (further explanation in section 2.3.3) is highlighted with a red-dotted line in Figure 5. Other portions of the standard ISO 26262 that are considered in this thesis are part 2 and part 8 (See Section [**?**]). The remaining parts of the safety automotive lifecycle are out of the scope of this master thesis.

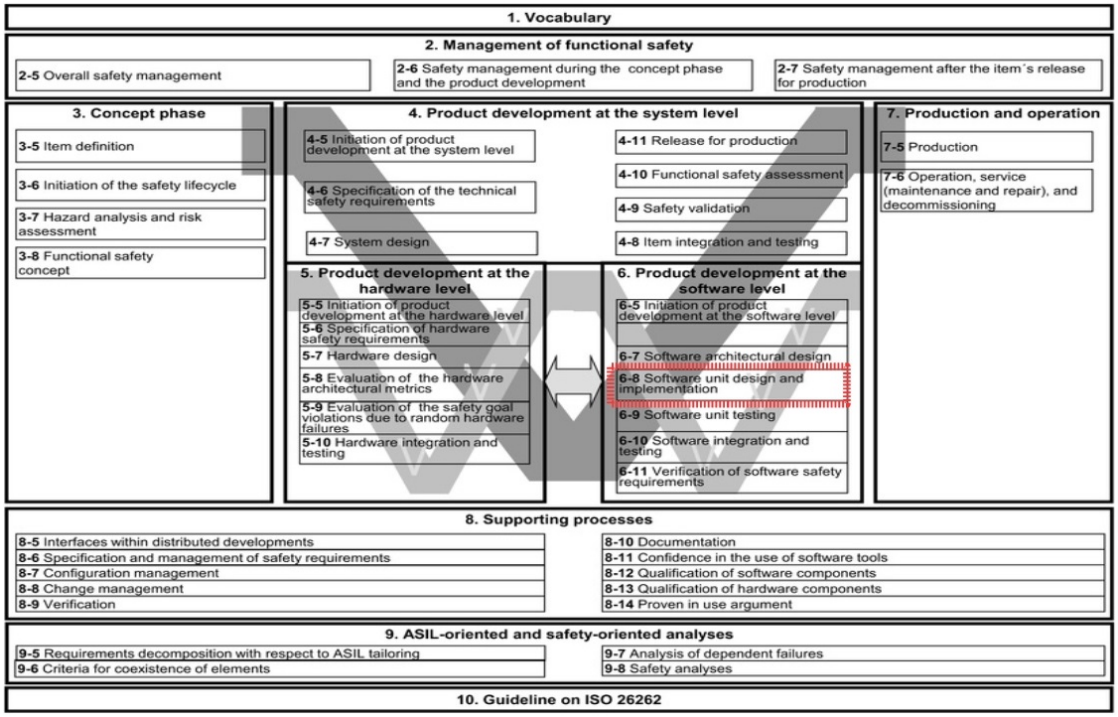


Figure 5: Overall structure of ISO 26262 [5].

2.3.3 Software unit design and implementation

Part 6 of ISO 26262 is entirely devoted to the specification of requirements for product development at the software level for automotive applications. The product development at the software level follows a process that recalls the V-model for software development (see Figure 6), where the activities on the left-hand side of the V-model pertain to the design phases and activities in the right-hand side of the V-model define the test phases of the software development. *The software unit design and implementation* is the bottom phase defined in the left-hand side of the V-model and is the object of the study of this present thesis. This phase is highlighted with a red-dotted line in Figure 6.

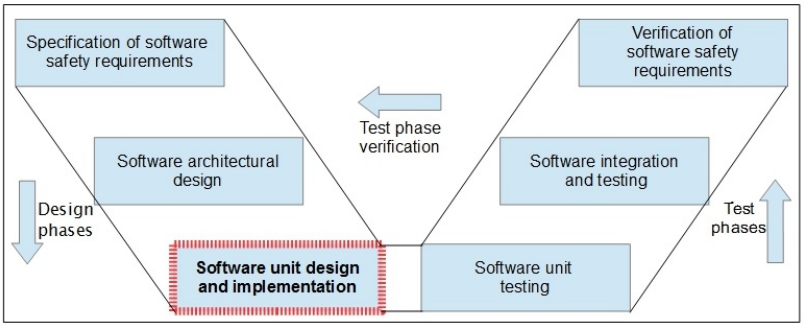


Figure 6: V-model for product development at the software level (adapted form [5]).

The software unit design and implementation phase aims at specifying the software units in

accordance with the architectural design (which is the resulting work product of the software architectural design phase) and the associated software safety requirements (work product defined in the specification of software safety requirements phase). This phase also addresses the implementation of the software units as specified. There are two work products resulting from this phase, namely *software unit design specification* and *software unit implementation*. These two work products are closely related, since for implementing a software unit, its design must be available. Additionally, these two work products are prerequisites of the software unit testing phase and subsequent phases of the ISO 26262 safety lifecycle.

ISO 26262-6 is very specific when describes the applicable requirements. The definition of the two work products of this phase is the result of the requirements specified in numerals *8.4.2* to *8.4.4* for the *software unit design specification*, and requirement *8.4.4* for the *software unit implementation*. These requirements can be summarized as follows: the internal design of software units in the software unit design specification has to be described to the level of detail required for its implementation. The specification of the software units shall describe the functional behavior as well. The software units can be implemented as a model or directly as source code, where prescribed modeling or coding guidelines are used. When implementing a software unit, both software safety requirements and non-safety-related requirements has to be included, so all the requirements are handled within one development process. Safety-related software units shall be complied with the software safety requirements. Besides, the software unit shall be described using the notations listed in Table 2.

Table 2: Notations for software unit design (adapted form [5]).

Notation	**A**	**B**	**C**	**D**
Natural language	++	++	++	++
Informal notations	++	++	+	+
Semi-formal notations	+	++	++	++
Formal notations	+	+	+	+

The notations listed in Table 2 are alternative notations, which means that an appropriate combination of these notations, in accordance with the ASIL indicated, can be applied. The notations in the table are also listed with different degrees of recommendation according to the ASIL. The levels of recommendation are marked as "++" and "+" in the table. There is a third recommendation level marked as "o", that is not presented in this table, but that is used in other requirements. So, the three recommendation levels are explained, according to the standard ISO 26262 [5], in the following way:

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

In Table 2 the column *Notation* is what in the standard is called *method*. Notations with higher recommendation e.g. "++", must be preferred during the description of the software units. If notations with lower recommendation are selected, e.g. those marked with "+", a rationale shall be given for explaining the decisions behind the selected combinations of notations. This rationale should demonstrate that the decisions comply with the corresponding requirement.

Source and object code are part of the implementation of the software units. Design and implementation related properties are achievable at the source code level, but if model-based development with automatic code generation is used, these properties apply to the model and are not necessary to be demonstrated at the source code level. The properties that shall be reached by the design and implementation of the software units are the following [5]:

1. *correct order of execution of subprograms and functions within the software units, based on the software architectural design;*

2. *consistency of the interfaces between the software units;*
3. *correctness of data flow and control flow between and within the software units;*
4. *simplicity;*
5. *readability and comprehensibility;*
6. *robustness;*
7. *suitability for software modification; and*
8. *testability.*

To achieve the properties previously listed, design principles for software unit design and implementation should be applied. The design principles are given in Table 3.

Table 3: Design principles for software unit design and implementation (adapted form [5]).

Principle	**A**	**B**	**C**	**D**
One entry and one exit point in subprograms and functions	++	++	++	++
No dynamic objects or variables, or else online test during their creation	+	++	++	++
Initialization of variables	++	++	++	++
No multiple use of variable names	+	++	++	++
Avoid global variables or else justify their usage	+	+	++	++
Limited use of pointers	o	+	+	++
No implicit type conversions	+	++	++	++
No hidden data flow or control flow	+	++	++	++
No unconditional jumps	++	++	++	++
No recursions	+	+	++	++

The design principles listed in Table 3 have to be taken into consideration in the design of a software unit, and a correct combination of them must be selected. Additionally, they have to be selected according to the highest level of recommendation for the ASIL stated, or otherwise, a rationale should be provided.

2.4 Documentation management

Documents are considered some of the major milestones in a project, because they drive the project, organize it, standardize it, and provide communication means between those implied in the project [38]. Documentation in software engineering captures in-depth knowledge about the software project and plays an important role in knowledge transition [39]. Therefore, documentation management is a critical activity in the realization of any project. When adopting ISO 26262, documentation management is very crucial, since one of the requirements is the creation of a safety case. This section presents information related to the documentation process when ISO 26262 is applied. The section is structured in the following way: Subsection 2.4.1 presents a safety case in the context of ISO 26262, and Subsection 2.4.2 presents the documentation management in the context of ISO 26262.

2.4.1 A safety case in the context of ISO 26262

In the context of ISO 26262, a safety case represents a significant work product. This work product is essential as it *"should progressively compile the work products that are generated during the safety lifecycle"* [5]. A safety case is composed of three main elements: *requirements*, which describe the objectives to obtain; *arguments*, that can be used to show the relation between different requirements; and *evidence*, which is used to support the requirements [3]. In a safety case, it is possible to find process-based or/and product-based arguments. When the argumentation relies on the development techniques, it is considered a *process-based argumentation*, and when

the arguments are prescribed by the generation and assurance of product-specific evidence that meets the safety requirements, it is seen as *product-based argumentation* [40]. Safety evidence, according to [41], can also take tree forms: *immediate evidence*, evidence which is itself evaluated like source code, *direct evidence*, which presents properties of the candidate, like test results, and *indirect evidence*, which describes the circumstances relevant to the creation of the candidate, like development process. The immediate evidence in ISO 26262 is provided by the left-hand side work products of the V-model (See Figure 6), direct safety evidence is provided by the right-hand side work products of the V-model [20], and indirect evidence is provided by the information compiled of the general process. The aim of this thesis is to make the foundations for a product-based safety argumentation with the compilation of immediate safety evidence for the development of software units.

2.4.2 Documentation management in the context of ISO 26262

When adopting ISO 26262, work products are generated during the lifecycle. Organizing work products require a careful documentation management for avoiding inconsistencies between them, or nonconformity with the standard requirements. Likewise, a strict documentation process helps in the construction of a flawless safety case. So, the documentation process is an important part for showing compliance, and it is closely related to the safety lifecycle [20]. Each stage of the lifecycle generates work products that are linked to work products created in previous stages. The work products are also prerequisites for later activities. In conclusion, many work products are resulting from the activities of the lifecycle, and all of them are connected in some way. Changes in one work product may affect others. For example, the generation of work products may create the need to go back in the process for updating work products created in previous stages. It is difficult to analyze work products state individually at any given time, especially when decomposition hierarchy of the overall product, the different phases of the development, or organizational responsibilities are implied [42]. This feature makes the documentation process an arduous and complicated activity [20]. Additionally, the documentation process is prescribed in the standard, which means that special requirements must be considered when documentation is presented for assessment purposes. For example the purpose of the documentation process, according to ISO 26262 is to make documentation available:

- *during each phase of the entire lifecycle for the effective completion of the phases and verification activities*
- *for the management of functional safety, and*
- *as an input to the functional safety assessment.*

The standard presents requirements, but *"flexibility is allowed if properly introduced"* [20]. Regarding documentation management, organizations can decide how this documentation management should be carried out, since the standard does not prescribe the methods or tools that should be used.

2.5 Traceability and interoperability

Traceability [19] is defined, in the context of system and software engineering, as *"the degree to which a relationship can be established between two or more products of the development process"*. The relationship between the work products in the safety lifecycle proposed by ISO 26262 is evident, since work products are prescribed as prerequisites of other work products. Additionally, the completion and maturity of the work products, as well as their quality is monitored through tracking process [42]. The need to establish traces between the work products provided during the stages of the safety lifecycle plays an important role in gaining confidence in the safe operation of the system under development. Moreover, the lack of understanding of the safety evidence traceability can result in improper evidence management and therefore certification risks [29], or negative safety assessment results.

A recent study made in [39] demonstrates that some of the documentation issues that are persistent through the software lifecycle are related to traceability problems, e.g. documents are not traceable to the changes made in code, or documents are out of date. The study also found that the significance of software documentation is determined by, among others, tools that enhance navigation, searching, and traceability of documents. The inclusion of software tools generates a new challenge: the interoperability between the tools (*"the ability of a system or a product to work with other systems"* [43]). Interoperability can be reached using semantic web techniques (more explanation in Section 2.7), the so-called semantic interoperability. Beyond the ability of two or more computer systems to exchange information, semantic interoperability is *"the ability to automatically interpret the information exchanged meaningfully and accurately to produce useful results as defined by the end users of both systems"* [44].

2.6 Semantic knowledge representation

Semantic knowledge representation [45] corresponds to the assumption that knowledge representation systems use semantic techniques to represent information in a machine-readable form. Semantic knowledge representation is used for computer systems to solve complex tasks. This section aims to give an overview of the aspects related to semantic knowledge representation, so the reader is familiar with this techniques. This section is structured in the following way: Subsection 2.6.1 presents essential terms required in the understanding of this section, Subsection 2.6.2 presents a basic approach to knowledge representation, Subsection 2.6.3 presents an overview of knowledge domain, and Subsection 2.6.4 presents the generalities of UML profile.

2.6.1 Basic terms

This subsection recalls some terms required for the understanding of the topic treated in this section. The terms are organized alphabetically.

Aboutness is the information of what something is about [45].

Bridge is a layering dependency between two domains, where one domain makes assumptions and other domains take those assumptions as requirements [46].

Concept is a representation of abstract, real-life or fictive things, or aspect of things [45].

Domain is an autonomous, real, hypothetical, or abstract world inhabited by a set of conceptual entities that behave according to characteristic rules and policies [47].

Metadata is data about data. Metadata can be also defined as structured data about an object that supports functions associated with the designated object [46].

Term linguistic representation of concepts, strings respectively [45].

2.6.2 Knowledge representation

There is not an agreement about the definition of knowledge representation [48]. However, one assumption of what knowledge representation is taken from [45], where it is defined as *"the subsumption of methods and techniques to represent the aboutness of information resources"*. The concept is better understood if the term *information space* is recalled. Information space is a collection of information resources that can take two forms: first, when information is homogeneous (in form and content) and stored in a centralized way (for example, library resources), and second, when information is heterogeneous and distributed over several repositories (the Internet is one example). When carrying out searches, an information space must provide the aboutness of a resource. Normally, the aboutness of an information resources is represented by one or more subject headings (also called *descriptors*). These descriptors are linked to the information about the resource, which is called *entities of knowledge representation*. In Figure 7 the descriptors are

the round circles that have names like *resource type*, *name* or *document related* and the entities of knowledge representation are the gray circles, namely *Software unit*, *Fuel Level Estimation Algorithm* or *AE201*. An information space that has a network-like structure (like in Figure 7) is called *knowledge structure* since the concepts are somehow related. Different knowledge structures can have different representations, making them heterogeneous. The heterogeneity can be addressed, creating intersystem relations. Intersystem relations are established when two or more structures can talk using the same vocabulary. These vocabularies are commonly known as metadata. With metadata defined to control de information, the knowledge structures have now a *knowledge representation*.

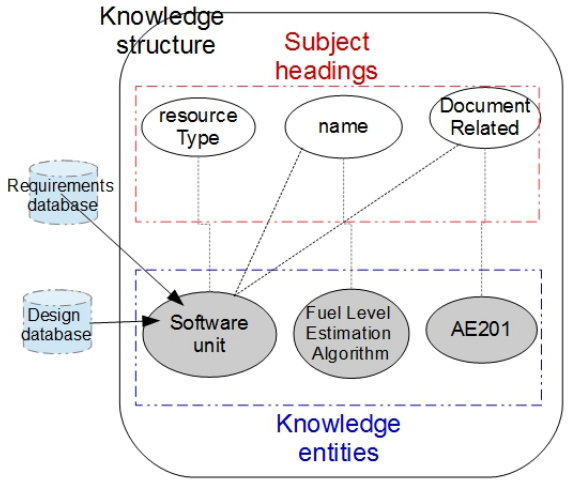


Figure 7: Example of an information space.

For exemplification purposes, a semantic information space is presented in Figure 7. Data used in this example is taken from Section 2.2.3, where it is stated that a software unit function, in the context of Scania, is called Fuel Level Estimation Algorithm and that a document related is called AE201.

2.6.3 Knowledge domain

To structure an information space, metadata can be used. This structure *entails that information is organized systematically* [46] and it is used to describe a property or a set of useful properties of an information resource [46]. To create a metadata structure, it is necessary to built restricted knowledge domains. A domain [47] is a semantically autonomous unit, i.e. they do not need the presence of other domains to be able to exist and to have meaning. However a domain is something that is part of a whole system. The main characteristics of a domains are [46]:

- *Domain mission: every domain has to have defined a purpose for existing. To be able to support this purpose (or mission) a set of related conceptual entities have to coexist.*
- *Domain autonomy: Each domain is autonomous. A conceptual item is defined once in each domain and relate to other conceptual items in the same domain. However, a conceptual item in one domain do not require the existence of other conceptual entities in other domains.*
- *Domain replacement: Being autonomous gives the opportunity to be replaceable.*

Domains are interconnected via bridges. The graphical representation of a domain and its bridges is called *domain chart*. In Figure 8 is depicted and example of a domain chart, where three domains called Architecture, Requirements and Test are related to their corresponding bridges.

In a domain chart, the domains are represented using the UML package and the bridges are dotted dependencies. Domain charts can be more formally represented by using UML Profile (explained in section 2.6.4). A domain chart only provides a name for each domain. For this, it

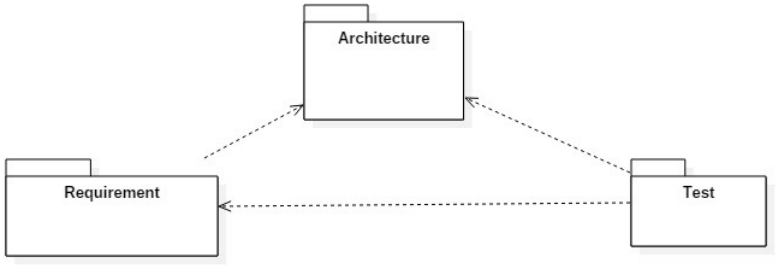


Figure 8: Simple representation of a domain chart.

is necessary to build a mission statement for each domain. Domain missions are very short, and their objective is to communicate the most relevant information. Assumptions to the appropriate bridge can also be defined. Every knowledge domain is composed of entities and terms, which are elements that come from the concepts we created about the *"things of the world"*. The elements of a knowledge domain are part of a metadata model. When the metadata models are encoded in a standardized machine-readable markup language, like RDF (explained in Section 2.7.3), they are considered metadata schemas.

2.6.4 UML Profile

A UML profile [49] is a set of elements that collectively specialise and tailor the UML metamodel for a specific domain or process. A UML profile is a set of extension mechanisms grouped in a UML package, stereotyped as <<*profile*>>. The UML metamodel defines classes as metaclasses. A stereotype is a special metaclass, stereotyped <<*stereotype*>>, that allows the extension of any metaclass with any meta-attributes, and make it more accurate using additional constraints. The constraints can be specified in two ways: after the meta-attributes or as a note. In Figure 9, there is a portion of a domain defined in a UML package called *SoftwareUnitRelatedConcepts* and stereotyped <<*profile*>>. One stereotype was defined and called *SoftwareUnit*. This stereotype has three meta-attributes: *resourceType*, *name* and *documentRelated*. There are four constraints defined, three of them are defined after the meta-attributes, and one is defined in a note. One constraint, for example, states that there is just one possible value for the of the meta-attribute resourceType, this value has to be a string, and it has to be SoftwareUnit (resourceType: String[1] = SoftwareUnit).

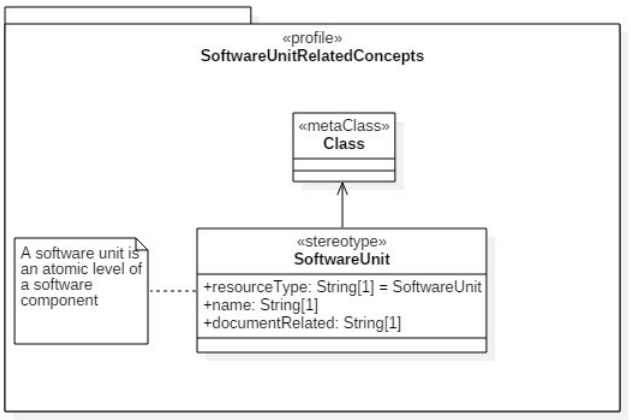


Figure 9: Simple representation of a UML profile.

2.7 Semantic Web fundamentals

This section aims to introduce basic information about Semantic Web and the languages associated. The languages are not recalled exhaustively, but simple examples concerning the purpose of the languages are provided and explained. The section is structured in the following way: Subsection 2.7.1 present the basic terms required for the comprehension of the overall Section, Subsection 2.7.2 presents the Semantic Web concept, subsection 2.7.3 presents the Resource Description Framework (RDF), Subsection 2.7.4 presents the RDF Schema, and Subsection 2.7.5 presents SPARQL.

2.7.1 Basic terms

Some terms that are important for the understanding of the overall semantic web concepts are recalled in this section. The terms are organized alphabetically.

IRI is a new protocol element, a complement to URIs. An IRI is a sequence of characters from the Universal Character Set (Unicode/ISO10646). There is a mapping from IRIs to URIs, which means that IRIs can be used instead of URIs where appropriate to identify resources [50].

Resource is anything that can have a URI [51].

Semantic reasoners are tools that can perform reasoning tasks, typically based on RDFS, OWL, or some rule engine. It is a subcategory of "Tool" [52].

Semantics represents, in the context of semantic web, the meaning of the web resources[53].

Serialization is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed [54].

Syntax represents, in the context of semantic web, the structure of the web resources, in other words, the data model [7].

TURTLE syntax means Terse RDF Triple Language and it is a format for expressing data in the Resource Description Framework (RDF) data model with a syntax that follows a triple patterns structure [55].

URI is a compact sequence of characters that identifies an abstract or physical resource [56]. URIs are used to access a specific object given a unique name or identifier. They provide a common syntax for naming a resource regardless of the protocol used to access the resource. URIs can include either a complete or a partial location. It can optionally include a fragment identifier separated from the URI by a pound sign (#).

Web browser is a software program that permits one to access the internet via web addresses.

Well-formed XML is an XML document with correct syntax. There are five syntax rules for a well-formed XML document: first, an XML document must have a root element; second, XML elements must have a closing tag; third, XML tags are case sensitive; fourth, XML elements must be properly nested; and fifth, XML attribute values must be quoted [57].

XML is a language designed to stored and transport data. Data defined in XML is both human- and machine-readable [57] and its structure is based on tags (definitions embraced in <>).

XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names [58]. A eXtensible Markup Language (XML) namespace is declared using the reserve XML attribute *xmlns*. A namespace has two attributes, the name and prefix. The namespace name is a URI reference which should be unique and persistent. The namespace prefix is used to associate elements and attributes names with the namespace name.

XML/RDF is a serialization type where XML language is used encode RDF information [59].

2.7.2 The Semantic Web

The Semantic Web [60], a Web mainly designed for automatic processing, is an extension of the conventional World Wide Web (WWW) (called simply Web) techniques, where machine-readable capabilities are designed and exploited. It is built using the principles and technologies of the Web, but enriched with the syntax provided by XML, and the semantics provided by RDF and RDF related languages [61]. The Semantic Web provides an environment where applications can, among other things, query data and draw inferences using vocabularies [62]. Semantic web reuses Web's global indexing to search and localize data, as well as naming scheme to represent every semantics concept (or resource). So, standard Web browser, as well as semantically aware applications (called semantic reasoners) can access the information provided by the semantic documents following their unique identifier (URI) [63]. The Semantic Web provides a common framework, where a collection of technologies and standards allows data to be shared and reused. Figure 10 shows the Semantic Web stack depicted during 2015, where a new layer called *RDF Data Shapes* is presented (highlighted in the figure with a red-dotted line). This layer will be explained in section 2.8. Following, RDF Model & Syntax, a layer where RDF language is located, RDF schema and SPARQL are briefly explained. The mentioned four layers are the main the focus of this thesis.

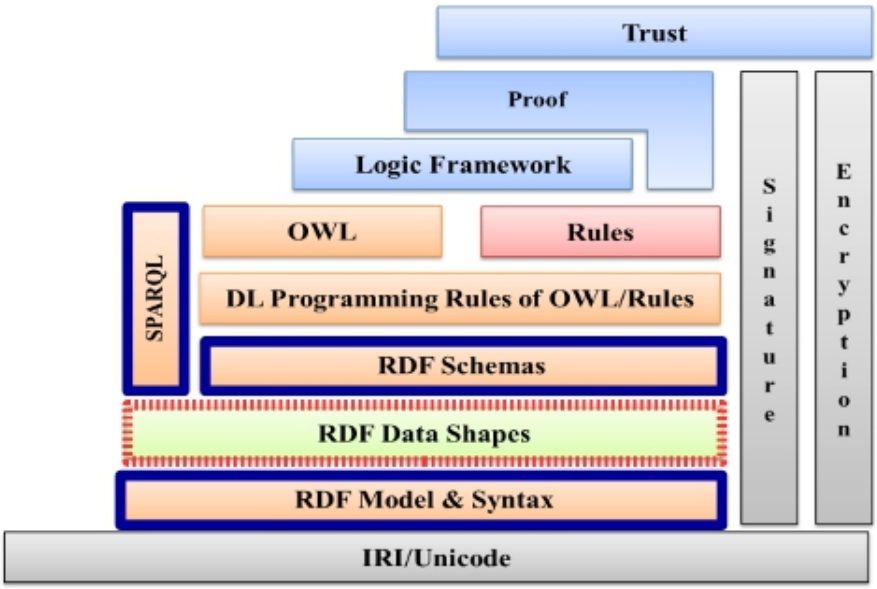


Figure 10: Semantic Web Stack (2015) [6].

2.7.3 Resource Description Framework (RDF)

RDF [7] is a framework for expressing information about resources on the Web. RDF provides *"means of recording data in a machine-understandable format, allowing for more efficient and sophisticated data interchange, searching, cataloging, navigation, classification and so on"* [64]. RDF resources are identified using web identifiers (URIs) and described in terms of simple properties and values [65]. The first recommended RDF specification was released in 1999 and a candidate recommendation for RDFS specification, which provides a data modeling vocabulary for RDF data [66], appeared in 2000. The RDF model is composed of two fundamental data structures [7]:

- *RDF graph: also called RDF triple. It is the minimum structure used to express descriptions of resources. It documents three pieces of information in a consistent manner, allowing both human and machine consumption of the same data. Figure 11 shows an RDF directed graph, a method used to describe RDF data models.*
- *RDF dataset: represents multiple RDF graphs. They allow working with multiple RDF graphs while keeping their content separate.*

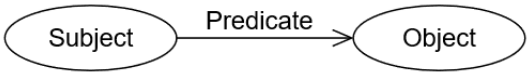


Figure 11: RDF graphs representation [7].

In RDF terms [7], the *subject* is the thing being described (a resource identified by a URI), the *predicate* is a property type of the resource, and the *object* is equivalent to the value of the resource property type for the specific subject. The core RDF vocabulary is defined in an XML namespace, commonly called *rdf*, where the URI is `http://www.w3.org/1999/02/22-rdf-syntax-ns#`.

The following commented XML/RDF-based fragment shows an example of an RDF resource, where a software unit resource is described.

```
<?xml version="1.0"?>
<!--rdf document that used the prefix rdf and ex (namespace selected for the
    example)-->
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:ex="http://example.com/elements/1.0/">

   <!--Definition of the resource software unit-->
   <ex:SoftwareUnit>
      <!--property identifier-->
      <ex:documentRelated>AE201</ex:documentRelated>
      <!--property name-->
      <ex:name>Fuel Level Estimation Algorithm </ex:name>
   </ex:SoftwareUnit>

</rdf:RDF>
```

Listing 1: software unit resource described in XML/RDF serialization.

There are RDF validation services (or tools), like the validator provided by W3C [3]. These kind of tools check that the XML/RDF documents have well-formed XML structure, and in some cases, provide the graphical structure of the RDF graphs. Figure 12 shows the graph of the piece of XML/RDF code presented in Listing 1. The graph is provided by the W3C validator.

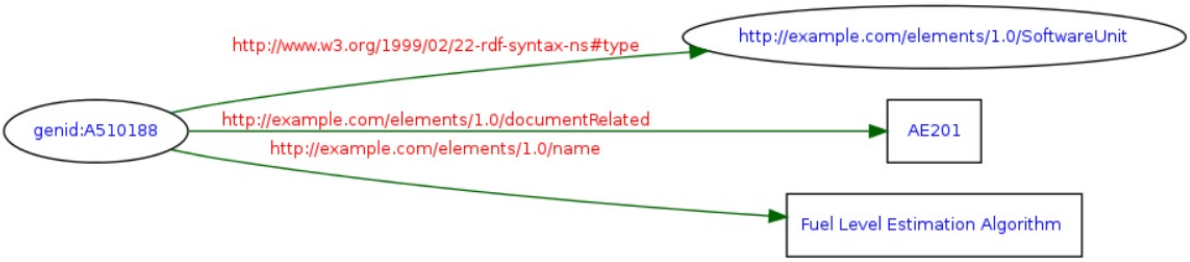


Figure 12: RDF graph of the XML/RDF example described in Listing 1.

The RDF dataset of the example is composed by four nodes. RDF nodes can have the following types [7]:

- **IRI**: provides a specific identifier unique to the node and it is drawn in the graph with a ellipse around the identifier. In the example, the IRI is `http://http://example.com/elements/1.0/SoftwareUnit`.

[3] https://www.w3.org/RDF/Validator/

- **Blank nodes**: are nodes that do not have any web identifier. In the graphs, they are shown as empty circles, but most RDF parsers and building tools generate a unique identifier for each blank node with the form *genid(unique identifier)*. In Figure 12, the blank node is generated by the RDF parser (a software application that can process RDF information) and labeled *genid:A510188*
- **Literals**: consist of three parts: a character string, and an optional language tag and data type. They represent RDF objects only, never subjects nor predicates. RDF literals are drawn with rectangles around them. In Figure 12, there are two literals: *"AE201"* and *"Fuel Level Estimation Algorithm"*.

The arcs in the RDF graphs represents the predicates. Predicates connect information, so they also have the form of an IRI. In Figure 12, the predicates are `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`, `http://example.com/elements/1.0/identifier` and `http://example.com/elements/1.0/name`

2.7.4 Resource Description Language Schema (RDFS)

RDF Schema [66], also called RDFS, is a semantic extension of RDF that *"provides a data-modeling vocabulary for RDF data"* [67]. RDFS is not required for an RDF document, but the schema approach *"guarantees that a particular RDF document is semantically and syntactically consistent across implementations"*[64]. RDFS defines which vocabulary elements are classes and which are properties. RDFS matches a property with a specific element as well as defines the range for each property. The type of literal that each property refers (string, number, and so on) can also be documented with RDFS. The RDFS class and property system are similar to those used in the Object Oriented Paradigm. The core vocabulary *"is defined in a namespace"* [66], commonly called rdfs, where the Uniform Resource Identifier (URI) is `http://www.w3.org/2000/01/rdf-schema#`.

Listing 2 shows a commented XML/RDF fragment with the RDFS vocabulary definition that appropriately matches the XML/RDF serialization shown in Listing 1.

```
<?xml version="1.0"?>
<!--rdf document that used the prefix rdf, rdfs and ex (namespace selected
    for the example)-->
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:ex="http://example.com/elements/1.0/">

   <!--Vocabulary definition: resource software unit-->
   <rdfs:Class rdf:about="http://example.com/elements/1.0/SoftwareUnit">
      <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#
          Resource"/>
   </rdfs:Class>

  <!--Vocabulary definition: property documentRelated-->
  <rdf:Property rdf:about="http://example.com/elements/1.0/documentRelated">
     <rdfs:subClassOf rdf:resource="http://example.com/elements/1.0/
         SoftwareUnit"/>
  </rdf:Property>

  <!--Vocabulary definition: property name-->
  <rdf:Property rdf:about="http://example.com/elements/1.0/name">
     <rdfs:subClassOf rdf:resource="http://example.com/elements/1.0/
         SoftwareUnit"/>
  </rdf:Property>

</rdf:RDF>
```

Listing 2: RDFS model in standard XML/RDF notation.

2.7.5 SPARQL

SPARQL [68] is a query language used to define queries on RDF data. The name is *"a recursive acronym for SPARQL protocol and RDF query language, which is described by a set of specifications from the W3C"* [69]. To extract data using SPARQL queries, conditions that reach the required information need to be defined. These conditions follow a triple pattern, which is similar to TURTLE syntax. Additionally, variables (defined using the symbol ?) are used in the queries to make them more flexible.

In Listing 3, an example of an SPARQL query is presented. This query is applied to the RDF information presented in Listing 1 and it aims to extract the name of the resource that has identifier *AE201*. The query gives as a result *Fuel Level Estimation*, which is the correct name stored in the software unit identified as *AE201*. The variables defined in this example are *?SoftwareUnit* and *?name*.

```
#Definition of the RDF data source
@prefix ex: <http://example.com/elements/1.0/>.

#Definition of the information required
SELECT ?name

#Defintion of the condition
WHERE {
    ?SoftwareUnit a ex:SoftwareUnit .
    ?SoftwareUnit ex:documentRelated "AE201" .
    ?SoftwareUnit ex:name ?name .
}
```

Listing 3: RDF model in standard XML notation.

2.8 RDF data shapes

This section presents the information related to the RDF data shapes layer presented in figure 10. It is structured in the following way: Subsection 2.8.1 presents the basic terms required for the understanding of this section, Subsection 2.8.2 presents an overview of the RDF data shapes layer, Subsection 2.8.3 presents Resource Shape (ReSh), Subsection 2.8.4 presents Shape Expressions (ShEx), and subsection 2.8.5 presents Shapes Constraint Language (SHACL).

2.8.1 Basic terms

Constraint is a rule, expressed as a calculation in terms of other classes, attributes, and associations, that restricts the values of the attributes and/or associations in a model [47].

Constraint language is a declarative language for describing and applying constraints in data models [9].

RDF property is a relation between subject resources and object resources [7].

SHEXc is a serialization structured as a triple, similar to RelaxNG[4], and schema language for XML [70].

2.8.2 RDF data shapes layer overview

The RDF Data Shapes layer (see Figure 10) constitutes an effort of the W3C RDF Data Shapes Working Group[5] to produce a W3C recommendation *"for describing structural constraints and validate RDF instance data against those"* [71]. The W3C RDF Data Shapes Working Group,

[4]http://relaxng.org/
[5]https://www.w3.org/2014/data-shapes/charter

a group that was created after the RDF Validation Workshop[6] organized by W3C in 2013, aims at finding a proper way for validating RDF data according to specified patterns. These patterns correspond to the requirements and uses cases collected in a purposely created database of RDF validation requirements[7]. There are already several constraint languages that can be used for validating RDF data. However, semantic web projects still lack common tools and methodologies to describe and validate data [72]. Therefore, there is not a favorite constraint language that can be used for this purpose. In the following sections the constraint languages ReSh, ShEx and SHACL, are studied. These languages have been developed to describe rules that apply to RDF models. One reason for using those languages in the context of this thesis is that they have been covered by the W3C umbrella in one or other way. Additionally, these languages are popular among data practitioners, so reliable sources of documentation exist.

2.8.3 Resource Shape (ReSh)

Resource Shape (ReSh) [8] is an RDF representation of a resource that describes and constraints the RDF representation of other resources. ReSh uses the terms defined by the OSLC vocabulary (OSLC is explained in section 2.9), as well as RDF vocabulary terms. Resource shapes, according to [73], *"provide a way for servers to programmatically communicate with clients the types of resources they handle and to validate the content they receive from clients"*. A resource shape describes the RDF resource with the set of RDF properties that are expected or required. A RDF property is represented as a triple, *"whose subject is the resource, the predicate is the property and the object is the value"* [74]. The value of a property may be constrained to take one of a group of allowed values. When the value is another resource, this resource may be provided with another resource shape. Figure 13 presents the main concepts and relations related to ReSh [8]. The two boxes on the left (and colored in blue) represent external resource types that use shapes. The three boxes that are more to the right (and colored in pink) are the specification of the resource shape.

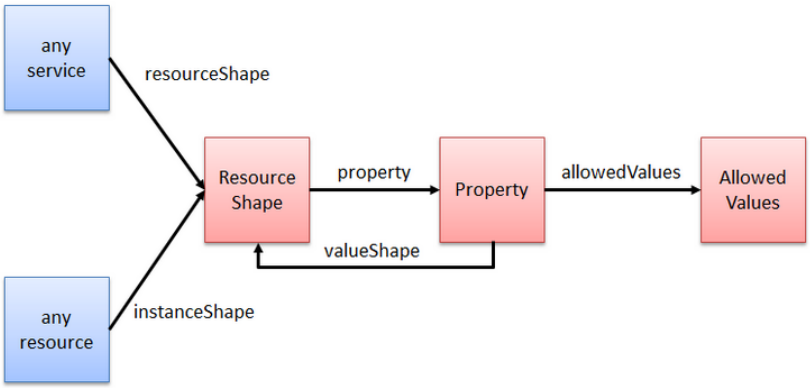


Figure 13: Diagram of main concepts and relations in ReSh [8].

An example of a resource shape is present Listing 4. The example shows the resource shape representation of the resource described in Listing 1, with one property: identifier. The property name can be defined in a similar way. The resource Shape uses OSLC definitions are further explained in section 2.9.

```
<?xml version="1.0"?>
<!--rdf document that used the prefix rdf, dcterms (namespace for Dublin
    Core vocabulary) and oslc (namespace defined for OSLC core)-->
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dcterms="http://purl.org/dc/terms/"
```

[6] https://www.w3.org/2012/12/rdf-val/
[7] http://lelystad.informatik.uni-mannheim.de/rdf-validation/

```
    xmlns:oslc="http://open-services.net/ns/core#">

<!--Resource Shape for Software Unit-->
<oslc:ResourceShape
    rdf:about="http://example.com/provider/shapes/SoftwareUnitShape">
    <!--Name of the resource shape created-->
    <dcterms:title>Software Unit Shape</dcterms:title>
    <!--Definition of the resource that is being described-->
    <oslc:describes rdf:resource="http://open-services.net/ns/example#
        SoftwareUnit"/>

    <!--Definition of the property identifier-->
    <oslc:property>
        <oslc:Property>
            <!--Name of the property-->
            <oslc:name>documentRelated</oslc:name>
            <!--Cardinality of the property-->
            <oslc:occurs rdf:resource="http://open-services.net/ns/core#Exactly
                -one"/>
            <!--Vocabulary definition-->
            <oslc:propertyDefinition rdf:resource="http://purl.org/dc/terms/
                identifier"/>
            <!--Data Type definition-->
            <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#
                string"/>
            <!--readOnly true, means that the property can be only change by
                the author-->
            <oslc:readOnly>true</oslc:readOnly>
        </oslc:Property>
    </oslc:property>

...
</oslc:ResourceShape>
</rdf:RDF>
```

Listing 4: Resource Shape example.

To express relationships between different resources, the recommendation given by [75] is *"to express a link as a property with a value-type of **Resource** and representation of **Reference**"*. For example a property *isRelatedTo* that may define the connection between two software units can be expressed in the way described in Listing 5.

```
    <!--relationship isRelatedTo-->
    <oslc:property>
        <oslc:Property>
            <!--Name of the property ( in this case, name of the relationship)
                -->
            <oslc:name>isRelatedTo</oslc:name>
            <!--Cardinality of the relationship->
            <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or
                -many"/>
            <!--Vocabulary definition of the property-->
            <oslc:propertyDefinition rdf:resource="http://open-services.net/ns/
                example#isRelatedTo"/>
            <!--To be able to connect two resources, valueType must be resource
                -->
            <oslc:valueType rdf:resource="http://open-services.net/ns/core#
                Resource "/>
            <!--To be able to connect two resources, representation must be
                reference-->
```

```
        <oslc:representation rdf:resource="http://open-services.net/ns/core
            #Reference"/>
        <!--The range is the resource that is connected with the
            relationship defined -->
        <oslc:range rdf:resource="http://open-services.net/ns/example#
            SoftwareUnit"/>
        <oslc:readOnly>true</oslc:readOnly>
      </oslc:Property>
    </oslc:property>
```

Listing 5: Relationship expressed with Resource Shape.

2.8.4 Shape Expressions (ShEx)

Shape Expressions (ShEx) [14] is a language that describes RDF graph structures, through a series of constraint rules (set of properties), written with a syntax called SHEXc. Rules described in Shape Expressions (ShEx) are used to identify predicates, their cardinalities, and datatypes, and they are useful for evaluating the nodes that are referred to the instance data. The rules can be formulated as conjunctions of constraints separated by commas and enclosed in brackets. For exemplification purposes, a piece of SHEXc notation is presented in Listing 6.

```
prefix oslc: <http://open-services.net/ns/core#>
prefix dcterms:<http://purl.org/dc/terms/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

start = <SoftwareUnitShape>

# Software Unit Shape
<SoftwareUnitShape> {
    dcterms:documentRelated xsd:string ,
    oslc:name xsd:string ,
}
```

Listing 6: Shape Expression of a software unit using SHEXc serialization.

The example in Listing 6 states that the properties identifier and name can be defined exactly once. If the cardinality of a property is defined differently, the expression cited in Table 4 has to be used. For example, if we would like that the resource described in the Listing 6 has more than one name, we should write the ShEx expression in the way described in Listing 7.

```
    oslc:name xsd:string+,
```

Listing 7: Property defined One-or-many times, using ShEx.

Expression	Meaning
	Exactly-one
?	Zero-or-one
+	One-or-many
*	Zero-or-many
{y,z}	x occurs minimum y and maximum z

Table 4: Cardinalities expressions for ShEx resources [14].

2.8.5 Shapes Constraint Language (SHACL)

Shapes Constraint Language (SHACL) [9] describes and constraints the content of RDF graphs (called here nodes), grouping constraints into "shapes". The shapes specify conditions that an RDF

node must follow. SHACL has its vocabulary, but it uses RDF and RDFS vocabulary, specially to define types, classes, subclasses, properties, lists, and resources. Some relationships between SHACL, RDF and RDFS are illustrated in Figure 14.

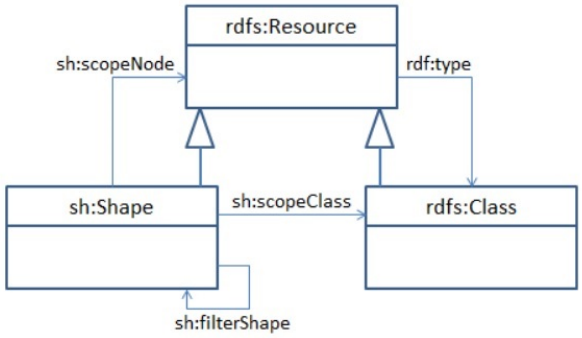


Figure 14: Illustration of some relationships between classes of SHACL, RDF and RDFS [9].

The core features of SHACL are the following [9] :

- **Shapes**: a group of constraints used to validate a node. Shapes are instances or the class *sh:shape*.
- **Scopes**: are used to linking nodes. There are three kinds of scopes: *node scopes (sh:scopeNode)* that link a shape with a specific node, *class-based scopes (sh:scopeClass)* which links all instances of a class with a shape, and *general scopes (sh:scope)* which is a flexible mechanism to link nodes in an arbitrary way.
- **Filter Shapes**: is a concept, *sh:filterShape* used to support use cases, acting as a precondition to the nodes before they are validated.
- **Constraints**: are different kind of restrictions that can be specified to resources. There are three types: *properties (sh:property)* which are constraints that define the restrictions on the values of a given property in the context of the focus node; *inverse properties (sh:inverseProperty)* that links a shape with constraints about a given property traversed in the inverse direction of the focus node; and *constraints (sh:constraint)* that link a shape with constraints that do not involve just a single dedicated property.

Listing 8 shows an SHACL shape fragment (serialized using TURTLE syntax) that expresses the properties formulated in XML/RDF serialization shown in Listing 1.

```
# baseURI
@prefix : <http://open-services.net/ns/iso26262am#> .

# prefixes
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc: <http://open-services.net/ns/core#> .
@prefix ex: <http://open-services.net/ns/example#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

# definition of the resource
ex:SoftwareUnit
  rdf:type rdfs:Class ;
  rdf:type sh:Shape ;
  rdfs:label "Software unit"^^xsd:string ;
```

```
  # definition of the property identifier
  sh:property [
      sh:datatype xsd:string ;
      sh:description "Document that is related to the software unit"^^
          xsd:string ;
      sh:maxCount 1 ;
      sh:minCount 1 ;
      sh:name "documentRelated"^^xsd:string ;
      sh:predicate dcterms:identifier ;

  # definition of the property name
  sh:property [
      sh:datatype xsd:string ;
      sh:description "Local Name of the software unit "^^xsd:string ;
      sh:maxCount 1 ;
      sh:minCount 1 ;
      sh:name "name"^^xsd:string ;
      sh:predicate oslc:name ;
    ] ;
.
```

Listing 8: Example of a SHACL shape.

In the XML/RDF fragment presented in Listing 8 a resource is defined with the types *rdf:Class* and *sh:Shape*. Every property is defined with constraints, like *sh:datatype* that constraint the data type of the property, *sh:description* that gives information about the property, *sh:maxCount* and *sh:minCount* which define the boundaries of the cardinality of the property, *sh:name* which is used to define the name of the property, and *sh:predicate* that describes the provenance of the vocabulary used for defining the property.

2.9 Open Services for Lifecycle Collaboration (OSLC)

This section presents general aspects related to OSLC, the principles in which it is based, and specific information required for the development of this thesis. This section is divided into four subsections: Subsection 2.9.1 presents Linked Data, Subsection 2.9.2 presents an overview of OSLC, Subsection 2.9.3 presents the OSLC Core specification, and Subsection 2.9.4 presents the OSLC property constraints. This section is mainly based on [10].

2.9.1 Linked Data

Linked Data [76] is a Semantic Web technique that recommends best practices for exposing, sharing and connecting resources, using Web technologies. With the use of Linked Data, the conventional Web (also called by Tim Berners-Lee, the Web of hypertext), can be configured in a different manner, allowing the discovery of new information (this new configuration is called the Web of Data). The principles of Linked data [60], are:

- *Use URIs as names for things.*
- *Use HTTP URIs so that people can look up those names.*
- *When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).*
- *Include links to other URIs, so that they can discover more things.*

The use of Linked Data principles allows the discovery of resources on The Web. When a resource is discovered by another resource, a representation of the resource (like a copy of it) is retrieved using internet protocols. This mechanism for retrieving information from the Web is called dereferencing URIs. A dereferenceable URI represents a resource (written in HTML or XML), describing the resource that the URI identifies.

2.9.2 Open Services for Lifecycle Collaboration (OSLC) overview

OSLC [77] is *"an open and scalable approach to lifecycles integration, that simplifies key integration scenarios across heterogeneous tools"* [78]. OSLC allows interoperability between independent software and product lifecycle tools, integrating their data and workflows in supporting end-to-end lifecycle processes, using standard methods. OSLC key features are:

- *adoption of Linked Data principles (see Section 2.9.1) for linking lifecycle data.*
- *adoption of RDF standards and key concepts: Graph data model, URI-based vocabulary and the serialization syntaxes,*
- *definition of standard rules and patterns for integrating lifecycle tools,*
- *definition of a common approach to performing resource creation, queries, and so on,*
- *definition of common resource properties and domain specifications.*

2.9.3 OSLC core specification

OSLC Core specification [10] specifies standard rules and patterns that all domain workgroups must adopt in their specification. The current version of an specification for OSLC Core, approved by OSLC community, is the version 2.0. The namespace defined for the OSLC Core specification is `xmlns:oslc="http://open-services.net/ns/core#"` and the prefix is *oslc*. The OSLC structure is represented in Figure 15. Two particular concepts, Resource Shape (explained in section 2.8.3) and Resource (an RDF resource that may have properties and may link to other resources) are highlighted in the figure with a red-dotted line since these elements are the main focus of this thesis.

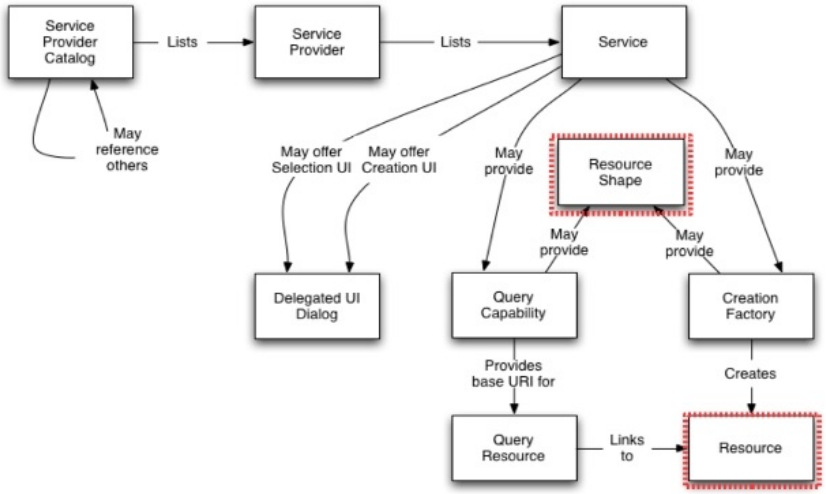


Figure 15: OSLC Core Specification concepts and relationships [10].

The core specification is used to create OSLC domains. An OSLC domain, according to [10], is a specific lifecycle topic that has its OSLC specification that complies with the Core Specification. Examples of lifecycle topics are change management or requirements management.

2.9.4 OSLC property constraints

An OSLC domain specification should provide a list of properties and their specific constraints. The elements required to formulate properties are [10]:

- *Name: Name of the property.*
- *Uniform Resource Locator (URL): The URI that identifies the property.*

- *Description: Description of the property.*
- *Occurs: Predefined values like Exactly-one, Zero-or-one, Zero-or-many or One-or-many.*
- *Value-types: Literal value types like Boolean, Date Time, Decimal, Double, Float, Integer, String, XML Literal or Resource value types like Resource, Local Resource, Any Resource.*
- *Representation: Properties with a Resource value type should also specify how the resource will be represented. The value can be reference, Inline or Either.*
- *Range: Used for properties with a resource-value type.*
- *Read-only: This value indicates whether or not clients are permitted to replace the property's value. Values can be true, false or unspecified.*

Table 5 shows the definition of the properties required by OSLC for the software unit specified in Listing 4 and Listing 5 .

Table 5: Properties defined in OSLC for the software unit.

Prefixed Name	**Occurs**	**Value Types**	**Rep.**	**Range**	**Description**
Properties: meta-attributes					
ex: documentRelated	Exactly one	String	n/a	n/a	Document that is related to the software unit.
oslc: name	Exactly one	String	n/a	n/a	Local name of the software unit.
Properties: relationships					
ex: isRelatedTo	Zero or many	Resource	Reference	ex: SoftwareUnit	Express a relationship between two software units.
Proposed IRI: `http://open-services.net/ns/ex#SoftwareUnit`					

2.10 Related work

The modeling of ISO 26262-compliant OSLC resources, where resources identification, representation, and shaping are presented together, is rather a new area of research. Not related work was found about the topic. However, approaches and methodologies, which cover the steps that are taken into account in the context of this master thesis, are not new in the research world. So the literature review carried out in this section is divided into three main streams: *metamodels for safety standards*, *tool integration frameworks*, and *RDF resource shaping*.

The idea of modeling standards is not new since models for safety standards are widely used for understanding and communication among engineers and software developers. So, in the group of research works related to **metamodels for safety standards** aimed at facilitating safety compliance is found *SafetyMet* [79]. SafetyMet is a unified metamodel in which all the models related to safety standards compliance must conform. The metamodel also includes concepts and relationships necessary for modeling and managing project-specific information. The metamodel presents several benefits, like the possibility of instantiation of all kind of safety models, the integration between them, and reuse of models. However is too generic, and, as the authors claim, it is necessary to specify more detailed attributes for the classes and further support in areas not addressed by the actual metamodel. This approach is valid and interesting, but can lead to confusion to those that do not have so much experience in safety assurance, because of the generality of the terms used in the metamodel. The approach presented in this current master thesis is towards the creation of a metamodel that suits the actual, specific domain terms of the standard ISO 26262, where the names of the classes presented in the metamodel are more closely related to the specific requirements presented in the standard. The approach presented in [80] presents a modeling approach for process process-based evidence, using specific terms from ISO 26262. The part of the standard used is the part 3, and traceability links between model

artifacts are clearly depicted. However, the product-based evidence is not modeled. The present master thesis is, instead, intended for targeting the product-based evidence metamodel. Specific product-based metamodels related to the standard ISO 26262 can be found in [81]. In this job, traceable work products for the software clauses corresponding to the right-hand side of the V-model in part 6 of ISO 26262 are represented as a collection of classes, attributes and relationship types. The modeling approach used in this master thesis is closely related to the job presented by [81] as it is its counterpart (the present thesis is creating a metamodel for the left-hand side of the V-model). The three works recalled aimed at creating metamodels for safety standards, where a plain collection of classes, attributes, and relationships, are depicted. They cover, in one or other way, the description of the domains in which they were conceived. However, the interest of the approach presented in the current master thesis is to propose the enrichment of the metamodel (created in the form of a UML profile) with the addition of constraints that guarantee the quality of the data annotated.

Tool integration frameworks using OSLC domain specification are also widely investigated. In [82] OSLC is used to integrate a smart production lifecycle. In this job, the integration scenarios created with OSLC could prove bidirectional interoperability. The project CRYSTAL (CRitical sYSTem engineering AcceLeration) is working on an interoperability specification based on OSLC, which aims at being an open European standard for the development of safety-critical embedded systems [83]. This specification should allow the interlinking of different kind of data across multiple tools, where work products related to the requirements management and the architecture management are defined. The adopting of lean enablers in systems engineering is proposed by [84], also using OSLC. This approach covers requirement, quality analysis, model-based systems engineering, model-based testing, product family engineering and safety analysis. The project's idea lies on fourth goals: improvement of the quality of system specifications, standardization of the development of variants that share commonalities, integration of the model-based system engineering with functional safety analysis, and the establishment of a tightly integrated systems engineering environment. In [6], a knowledge management specification for OSLC resources is introduced. Data tools are described and shared using also OSLC specifications in [85]. All these approaches made use of OSLC, generating specifications and tools chain for integrating environments. However, any of the approaches address, specifically, ISO 26262 safety lifecycle, which is the purpose of this master thesis. Besides, the OSLC domain specifications created by the OSLC community does not present domain-specific terms that tackle the granularity required for the definition of ISO 26262-compliant resources. This thesis aims at creating and proposing a fine-grain OSLC specification, that can be adopted by other ISO 26262 users.

OSLC approach relies on RDF standard for reaching a common interoperability framework. With RDF and XML in general, flexibility in the definition of resources can be achieved. This flexibility that can be a problem when integration activities are carried out, since the data quality, in most of the cases, is not evaluated. The use of ReSh, as the OSLC standardized data shaping specification, *"allows collaborating tools to check each other's expectations on resources"* [86]. However, the use of OSLC ReSh has its limitations. So, evaluations that tackle **RDF resource shaping** have been carried out. For this, a list of RDF constraints has been defined in [87] and [72], among others, to be evaluated by several RDF constraint languages. In [88] RDF constraints are evaluated in ReSh, ShEx, Web Ontology Language (OWL) and Description Set Profiles (DSP), giving not a favorite language for constraint the RDF resources, but outlining the idea of a better framework for supporting the mapping of high-level constraint languages. In [89], a survey of RDF constraint languages was done. This comparative study involved four RDF languages e.g. SPARQL Inference language (SPIN), ShEx, Stardog ICV and ReSh. The work presented in the current thesis aims at reviewing, in the first place, if the amount of RDF constraints already defined in [87] and [72] are enough for representing the ISO 26262 requirements or if this list has to be enhanced with more accurate ISO 26262-like requirements. On the other hand, there is a new RDF constraint language, called SHACL that has to be evaluated. In [90], there is a comparative study between SHACL and ShEx, where expressiveness of both languages is assessed, and possible translation between the two languages is analyzed. There is not a comparative study that includes the languages ReSh, ShEx, and SHACL, and the current thesis proposes this comparison, using the constraints found in ISO 26262.

3 Scientific research method

The research addressed by this thesis had as objective the establishment of an OSLC domain that targets ISO 26262 standard in an industrial context, specifically Scania AB. For reaching this goal was necessary to identify several elements. The first elements were related to the normative parts of ISO 26262, other elements were connected with the OSLC specification characteristics, and the last elements were affiliated with the understanding of the practices carried out at Scania. As part of the research, the study of how these three elements could be mixed to get a common interoperability schema was performed. Based on the nature of the research problem, the research methodology selected for carrying out this study was the qualitative research approach. This method, which according to [11] *"has a process that involves emerging questions and procedures, data typically collected in the participant's setting, data analysis inductively building from particular to general themes, and the researcher making interpretations of the meaning of the data"*, presents characteristics that suited better with the needs and the objectives of this thesis. In the context of the qualitative research, a case study was also conducted, so real data from the industrial setting were modeled. In Scania, safety process in the sense proposed by ISO 26262 are not carried out yet, since the current version of the standard (the one released in 2011) is not covering the kind of vehicles the company developed. However, the traditional methodologies used in the company for designing and implementing software units were used in the context of this thesis. The research method selected is based on Creswell's approach [11] and adapted according to the needs of the project (see Figure 16).

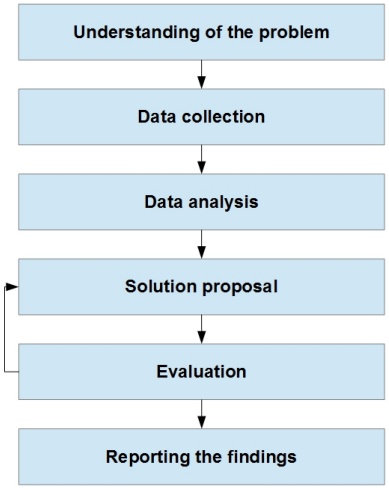


Figure 16: The research methodology used in the context of this thesis (adaptation from [11]).

The first part of the research corresponded with the understanding of the project's problem. Initially this step included the understanding of the context of the project, involving the OSLC specification, the ISO 26262 standard and Scania processes. The data collection was based on documents and previous experiences gathered in master thesis carried out at Scania. The written material mainly included in this research are standards, research papers, and Web pages. During the analysis phase, the interoperability of several OSLC domains was reviewed to identify their features. A comparative study of three RDF constraint languages was also carried out, so one could be selected in the final solution. The identification of the correct resources to be modeled and the constraints for guarantying the quality of data were also performed during the analysis phase. The key features of the standard ISO 26262, as well as the Scania practices, were included in this analysis. Partial solutions for the problem were regularly provided, during the solution proposal. These solutions were evaluated on a regular basis in order to refine the final solution. Once the final solution was found, A case study was applied, so the feasibility of the final solution could be established. The data obtained during the execution of the case study was subjected to a validation process that involves requirements engineering validation methodologies. The overall process was documented in the form of a thesis report.

4 Problem formulation and analysis

In this chapter, the problem to be solved is described in detail and analyzed in order to identify sub-problems. This chapter is organized as follows: In Section 4.1 the problem to be solved is presented in detail; and in Section 4.2 the problem is analyzed and decomposed into sub-problems.

4.1 Problem formulation

In Section 1.1 it was stated that to develop products in compliance with the standard ISO 26262, a safety case must be provided [20]. For creating a safety case, the safety process proposed by the standard must be adopted, and the safety evidence should be compiled [17]. The safety evidence, as recalled in Section 2.4.1, is composed of the work products resulting from every stage of the safety lifecyle [5]. The work products have to show, in a traceable way, that during the lifecycle, hazards have been identified, classified and appropriately mitigated. To be able to collect all the information presented in the work products, a careful documentation management, which is prescribed by the standard, has to be carried out. This prescription establishes requirements that have to be followed when the documentation management is performed. Requirements state that the documentation process should be planned in a way that the evidence is compiled progressively, the documents are clear and available when they are needed, and the search for relevant information is facilitated. All these characteristics make the documentation management an important but challenging activity, in the context of ISO 26262.

It is likely that Scania soon will be involved in the process of compliance with ISO 26262 [3, 13], a process that can contribute to the quality of the products elaborated, but also with the workload of the company [91], due to documentation process. The safety process can be softly introduced in the development process, to minimize this workload, if tools that help in the documentation management are included. These tools have two goals: first, they should take care of the information that is expected to be generated in the process of compliance with the standard, and second, they should address the traceability between all the documents so this information can be shared between those involved in the process. The introduction of tools can generate two new problems that have to be solved: first, tools are not actually adapted for managing ISO 26262-compliant-information, and second the information information generated by the tools can be inconsistent, due to lack of standard methods for sharing data. In other words, there is an absence of ISO 26262-compliant seamless interoperability.

4.2 Problem analysis

The main problem formulated in Section 4.1 is decomposed in small subproblems:

1. **OSLC domain specification development**: Every work product in the ISO 26262 lifecycle has specifically related information so that specific knowledge domains can be modeled. Every knowledge domain will be the base for building the controlled vocabulary required for the OSLC domain specification required. To address the creation of the domain specification, the following questions has been addressed:
    - **What are the elements involved in the definition of the OSLC domain specification?** The first step in the creation of an OSLC domain specification is the definition of the elements that should be included in the modeling of the domain.
    - **What is the existing OSLC domain specification that better suits the needs for the creation of ISO 26262-compliant resources?** There are already several domain specifications created in the OSLC community. These domains have been created with similar purposes to the one (or ones) required to be specified in this thesis. For this, an investigation of the existing OSLC domains specifications is needed, so domains with similar characteristics are extended, avoiding the creation of redundant domains. However, it can be possible that existing domains do not fulfill the needs of the domain required, so a new domain must be introduced.
    - **What are the existing vocabularies that can be used in the definition of an OSLC domain specification?** For creating an OSLC domain specification, metadata

vocabularies are reused. So, it is needed to recall terms already created in other vocabularies, to reuse them and make the domain created interoperable with other sources of information.

- **Which approach should be used for modeling the OSLC domain specification?** To support the modelling and ensure its repeatability, the approach to use has to be established.

2. **Comparative study on existing RDF data shapes languages**. OSLC resources are based on RDF. There are several RDF data shapes languages that can be used to constrain the data recorded as RDF resources. For defining the best option to use, the following questions have to be answered:

   - **What are the most common options available for shaping OSLC resources?** The OSLC community has defined OSLC ReSh for shaping OSLC resources. However, OSLC resources are defined in RDF, so other options for shaping the resources can be explored. These other options are called RDF data shapes languages and there many of them at a disposal.
   - **What are the technical characteristics of the RDF constraint languages selected?** Investigate the most important characteristics provided by the most common RDF data shapes languages are required in the definition of a language candidate for shaping the OSLC resources.

5 Methods to solve the problem

This chapter presents methods to solve the problem. These methods are found after answering the questions raised in Section 4.2. The methods defined in the current Chapter are then used for searching the solution in Chapter 6. The chapter is organized as follows: Section 5.1 presents the OSLC domain specification development, and Section 5.2 presents a comparative study of existing RDF constraint languages.

5.1 OSLC domain specification development

This section answers questions related to the OSLC domain specification development. It is organized in the following way: Subsection 5.1.1 presents the elements that are involved in the definition of the domain specification, Subsection 5.1.2 defines if the domain specification can be supported with the use of already defined specifications in OSLC, Subsection 5.1.3 presents the RDF vocabularies that are needed to support the development of the domain specification, and Subsection 5.1.4 presents the approach to use for modeling the domain specification in OSLC.

5.1.1 What are the elements involved in the definition of the domain specification?

Three fundamental elements are involved in the definition of an OSLC domain specification for software unit design and specification according to International Standarization Organization (ISO) 26262 and Scania context. The first element corresponds with the portion of the standard ISO 26262 that was selected for the study: part 6 - clause 8 (ISO 26262:6-clause 8 is explained in section 2.3.3). The purpose of this part of the standard is to guide the detailed design of the software unit and its subsequent implementation. The second element corresponds with the Scania practices related to the software unit design and implementation (the case study was introduced in Section 2.2.3). These practices are critical in the definition of the domain since the industrial context provided by Scania will contribute with the determination of the suitability of the application of the standard in the company and the appropriation of the solution provided. The last element corresponds with the understanding of the framework that was selected as a potential solution for addressing the traceability of the work products generated in this phase of the ISO 26262: OSLC (OSLC is explained in Section 2.9). OSLC has defined, with the help of specialized workgroups, several domain specifications that can be used for integrating lifecycle tools. There are three possibilities for integrating lifecycles using OSLC: The fist one consists in using the domain specifications that are already defined in the community without making any change in the specification. The second one consists in extending a domain specification, adding new terms. The last one consists in proposing a new domain specification that addresses lifecycles, or portions of lifecycles, that have not been defined yet. Subsection 5.1.2 clarifies the option that better suits the objectives of this thesis.

5.1.2 What is the existing OSLC domain specification that better suits the needs for the creation of ISO 26262-compliant resources?

To define the OSLC domain specification that better suits the needs for the creation of ISO 26262-compliant resources requires an analysis of the OSLC domains provided by the OSLC community. For this analysis, a study of the development status of the mentioned domains is required. The list of available status for the OSLC domain specifications, and their explanations is presented below [77]:

- ***Inactive****: there is no further development planned.*
- ***Scope****: the workgroup proposes, documents and prioritizes the scenarios and technical objectives that will be addressed in the current version.*
- ***Draft****: workgroup members comment on a contribute to a proposed specification through a series of drafts.*
- ***Candidate recommendation****: the specification is under review and open for comments.*

- ***Converge**: the broader community reviews and comments on the draft specification. Meanwhile, implementation and prototypes are initiated.*
- ***Finalize**: the steering committee reviews the specification and the specification undergoes a final polish to correct errors an unclear language.*
- ***Final**: the steering committee approves the final form of the specification. A final specification must have a working implementation and a test suite.*
- ***W3C recommendation**: the W3C fully endorses the specification.*

The domain specifications marked with the status *final* and *W3C recommendation*, listed in Table 6, are the ones that can be used in the creation of integrated scenarios.

Table 6: OSLC domain specification with final and W3C recommendation status.

Domain Name	**Coverage**
Core 2.0	Sets out the common features that every OSLC Service can be expected to support using terminology from the World Wide Web Consortium (W3C).
Architecture Management 2.0	Modeling, diagrams, and use cases for software development.
Assess Management 2.0	Reusable components, documentation, and representations.
Automation 2.0	Plans, requests, and results for builds and deployments.
Change Management 2.0	Defects, enhancements, changes, and tasks.
Performance monitoring 2.0	Availability, performance, and capacity.
Quality Management 2.0	Test plans, cases, and results.
Reconciliation 2.0	Refers to the case where there exists a need to understand if multiple providers are referring to the same resource, particularly when there is not already a common identifier that all providers populate.
Requirements Management 2.0	Stakeholder needs and how to meet them.
Linked data platform 1.0	Simple read-write Linked Data architecture using HTTP access and RDF data.

The domain specification Core 2.0 contains the basis for defining OSLC specifications. Moreover, OSLC domain specifications are expected to be built on top of this specification, since it *"establishes terminology and rules for defining resources in terms of the property names and value-types that are allowed and required"* [10]. So this specification will be present in the definition of the domain that is the focus of this thesis.

The name and the coverage of the domain specifications clearly explain what lifecycles are addressed. From subsection 5.1.2 it was understood that the phases of the lifecycle required to be studied correspond with one of the design phases of the V-model. There is not a domain in OSLC which name is equivalent to design phase (a domain called, for example, design management). However, design phases in software development are typically addressed, using modeling, diagrams and uses cases. These elements are covered by the domain specification called *Architecture Management 2.0*, so this specification becomes a good candidate to be used or to be extended. The other domain specifications can not provide elements for the domain specification that is the focus of this thesis.

The terminology of the domain specification *Architecture Management 2.0* is reduced to eight concepts, namely: *resource, architecture management resource, link, link type, link type resource, service provider, service description resource and service description document.* This approach was taken by the creators of the domain for avoiding the re-definition of *"model storage formats or even model or other architecture management resource notations"* [92]. However, the general overview of the software unit design specification given in Section 2.3.3, shows that a more precise terminology is required for modeling the ISO 26262-compliant work products. So, the candidate domain specification Architecture Management 2.0 is not suitable to be used or extended in the

case required in the context of this thesis. For this reason, a new domain specification, that tackles Architecture Management (AM) in the ISO 26262, has to be created from scratch.

5.1.3 What are the existing vocabularies that can be used in the definition of a the domain specification?

For creating new domain specifications in OSLC, and in general, when Linked Data new deployments are ongoing, one of the most difficult tasks is the determination of the RDF vocabularies that better suit the needs for describing the semantics of data. RDF vocabularies are themselves expressed and published using the Linked Data and Semantic Web principles, and there is already a well-documented ecosystem of Linked Data vocabularies managed by the Link Open Vocabularies (LOV) initiative [15]. These vocabularies are a collection of classes, properties, and data types that define the meaning of data, in specific contexts, giving humans and machines access to the terms used to qualify the data. The LOV dataset consists of 560 vocabularies as of July 2016, and it is constantly growing. A graphical representation of the vocabularies managed in LOV is presented in Figure 17.

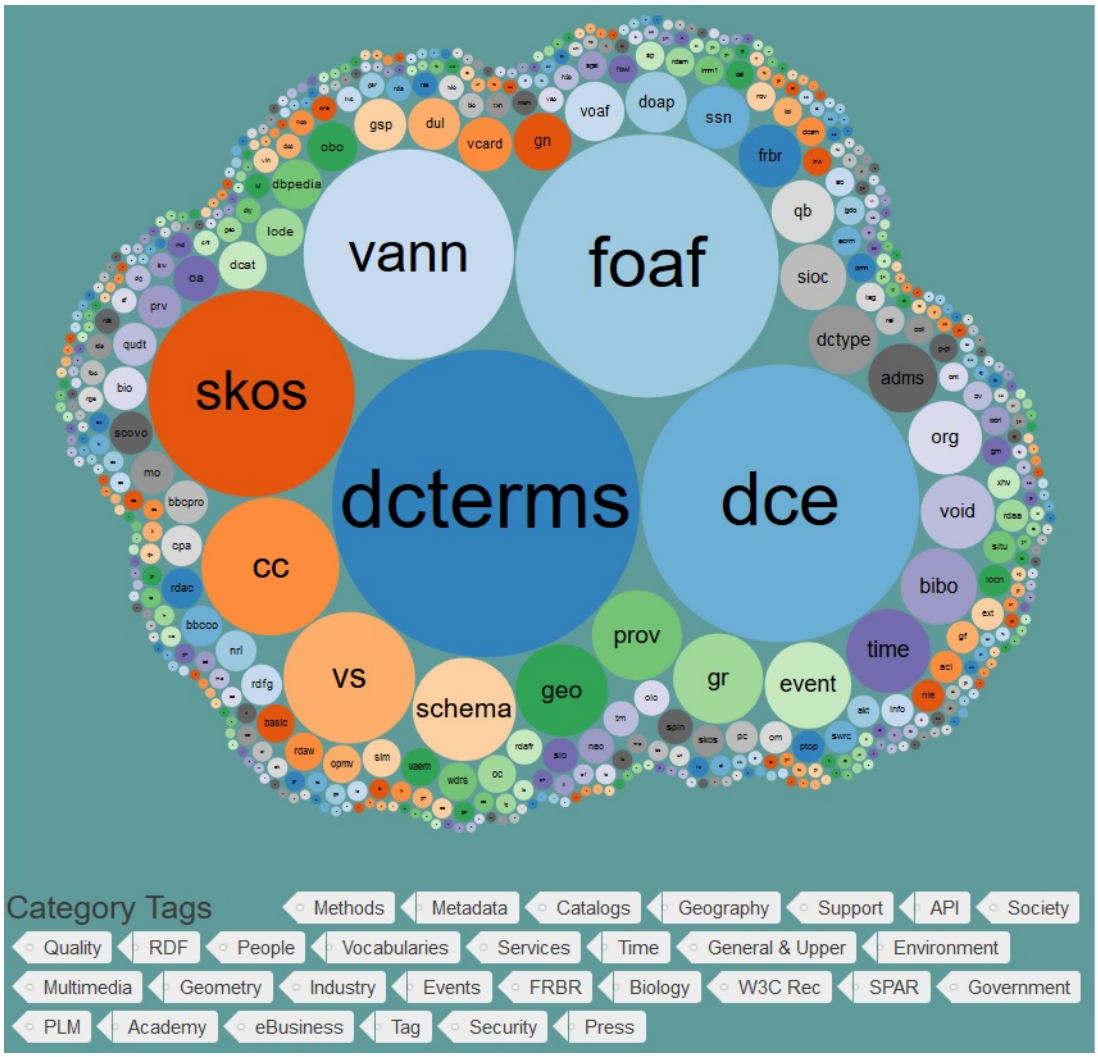


Figure 17: Link Open Vocabularies (LOV)[12].

The Vocabulary catalog provided by LOV is extensive, and even though is well organized (categories are explicitly presented in the diagram as they are marked as *category tags*), a profound study of the vocabularies provided there requires time. So, an in-depth investigation of the vocabularies, their purposes, and their terms is proposed as a future work. So far, a search of terms,

using the search engine defined in the LOV web page was used, to look for terms related with ISO 26262. There were no vocabularies or terms specifically related to the standard so, for the purpose of this thesis, it is assumed that new terminology is required. However, there are basic vocabularies that are mandatory in the definition of RDF vocabularies. Those, as well as the ones characterized in other OSLC domain specifications, are listed in Table 7. These vocabularies are the ones selected in the definition of the namespace declaration of the domain specification for software unit design and specification.

Table 7: Vocabularies selected form Metadata category [15].

Vocabulary	**Prefix**	**URI**	**Description**
Resource Description Framework	rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	Contains the elements required for defininf RDF resources.
Dublin Core Metadata metadata terms	dcterms	`http://purl.org/dc/terms/`	Metadata terms maintained by the Dublin Core Metadata Initiative, used for describing resources.
Friend of a friend	foaf	`http://http://xmlns.com/foaf/0.1/`	Based schema to describe persons and their social network in a semantic way.
RDF schema	rdfs	`http://www.w3.org/2000/01/rdf-schema#`	Provides a data-modelling vocabulary for RDF data. RDF Schema is an extension of the basic RDF vocabulary.
OSLC Core	oslc	`http://open-services.net/ns/core#`	The basic patterns and protocols that any OSLC software must implement.

5.1.4 Which approach should be used for modeling the domain specification?

The approach defined in [81] which aims at representing the compilation of traceable work products for the software testing clauses in part 6 of ISO 26262 was adapted and used for modeling the work products of this master thesis. The approach is summarized in the following steps: the work products required to be represented are analyzed and depicted in a metamodel, which is also instantiated (in the form of RDF resources) with specific information collected in an industrial setting, by the application of a case study. The case study was conducted at Scania, in three consecutive cycles. The first cycle was the definition of the conceptual metamodel, by analyzing documents e.g. ISO 26262 standard and related documents. The second cycle aimed to extend the metamodel with the information gathered in interviews with the people involved in the testing process at Scania. The third cycle addresses the validation of the extended metamodel. This methodology is adapted for the formulation of a metamodel that addresses the work products required by the clause software unit design and specification. Additional steps are added to the methodology since constraints required for enhancing the quality of data are also part of the present master thesis. This approach is summarized as follows:

1. **Gathering the domain-related data**: Information related to the domain of the interest is collected and organized. This data can be found in the form of documents or through informal interviews. The objective of this phase is to prepare the ground for the identification of the terms used in the domain. In other words, to gain an understanding of the requirements for the domain. The main sources of information are the latest published version of the standard ISO 26262 available at the company, classified internal documents, and other master thesis works carried out at Scania.

2. **Creating the metamodel structure**: The objective of this phase is to begin the abstraction of the information gathered into a model. This model should formalize the knowledge about the domain and should express the solution that meets the requirements of the portion

of the standard selected. Many formalisms that can be used to model the domain. However, the one selected is the UML profile, since this is the modeling approach followed in Scania, and this model should be aligned with the practices carried out by the company. UML profile, as explained in section 2.6.4, is an abstract, object-oriented formalism, which employs meta-classes, meta-attributes, and other object-oriented constructs as a way to organize abstractions. The identification of the elements required for the profile is performed in two ways:

(a) Modeling regulatory requirements: Elements are gathered from the standard ISO 26262-2011 (The current version available).

(b) Modeling company practices: The company practices are investigated, through the reading of documents.

3. **Attaching constraints to the metamodel**: Constraints enhance the semantics of the domain. They may be used to direct and optimize the structure of the implementation. To find constraints that can increase the quality of the data, a second review of the standard and documents related is carried out.

4. **Adding context to the domain specified**: According to [47], domains are determined for being part of a bigger context. So, the definition of information that gives context to the domain is carried out in this section.

5.2 Comparative study on existing RDF constraint languages

A domain specification can be materialized in a standardized data model, like RDF. RDF, as explained in Section 2.7.3, has the required flexibility for allowing the definition of the terms needed in a data model, and it is supported by the stack of maturing Semantic Web technologies (see Figure 10). However, ways for verification of the data are not as standardized as is desired. Verification of the data is required in a data model, so ambiguity in the information is avoided. This verification can be reached by the definition and delimitation of acceptable RDF data structure and content. Several RDF constraint languages have been created for this purpose, but there is a lack of agreement on which of them is more or less adequate. This section seeks to find options for constraint the RDF graphs, through the selection of RDF data shapes languages that better suits the needs of the domains specification that is the focus of this master thesis. This section is organized in the following way: Subsection 5.2.1 presents an explanation of what are the RDF shape constraint languages available for shaping resources, and Subsection 5.2.2 recalls in some technical characteristics that are presented in the RDF shape constraint languages selected.

5.2.1 What are the options available for shaping OSLC resources?

Data models like XML and RDF have already defined standardized schema languages (XML schema and RDFS, respectively). However, XML schema only provides syntactic interoperability but not semantic, and RDFS is just a vocabulary for describing resources [89]. So, these languages have limited capacity for restricting the possible values that an RDF graphs can admit. OWL, is an ontology language designed to represent knowledge about things, constrain this knowledge, and infer new knowledge based on predefined rules [93]. However, the use of OWL is limited because of the existence of two underlying assumptions: Open World Assumption (OWA) and Non-Unique Name Assumption (NUNA). OWA refers the assumption that the truth-value of a statement may be true irrespective of whether or not it is known to be true. NUNA refers to the assumption that different names always refer to different entities in the world [89]. Based on OWL, Stardog ICV is a system for validating Semantic Web and Linked Data [94]. Stardog ICV proposes integrity constraints modeled as OWL axioms, validating constraints, similar to checking database integrity constraints. It enables an OWL ontology to be interpreted as a set of integrity constraints checks that must be satisfied by the information explicitly presented or the information that may be inferred. The special characteristic provided by this language is that *"explicit information is needed to satisfy the integrity constraint"* [95]. So, the validation is closely related to the type of information given by the data. This feature makes the Stardog ICV a language not suitable

for constraint RDF graphs. SPARQL and SPIN are widely used for constraint formulation and validation, but constraints formulated in SPARQL *"are not as understandable as one wished them to be"* [87]. Moreover, SPIN is considered a low-level language in contrast to other constraint languages *"where specific language constructs exist to define constraints in a declarative and in comparison, more intuitive way"* [72].

According to [53], RDF and RDFS are *"unsophisticated and inexpressive"*, so *"there are not reasoning difficulties if all information comes in the form of RDF triples interpreted under RDFS semantics"* [95]. RDF triples can be treated as complete descriptors of the world where if a triple is not presented, then it is false (Close World assumption - CWA) and that different IRIs denote different individuals (Unique Name Assumption UNA) [95]. However, when users can understand the content of RDF datasets, SPARQL queries are less complicated to formulate. So, a description of the RDF datasets is needed. The Semantic Web community is aware of this shortcoming, and efforts for defining better RDF constraint languages have been carried out, since the realization of the RDF validation workshop in 2013[8]. For the purpose of this thesis three RDF data shapes languages are selected for being studied: Resource Shape (ReSh), introduced in Section 2.8.3; Shape Expressions (ShEx), introduced in Section 2.8.4; and Shapes Constraint Language (SHACL), introduced in Section 2.8.5. The reasons underlying the selection of these three languages is explained as follows: ReSh is selected because is the official language for shaping OSLC resources, ShEx is chosen because there are many research works (some examples are [89], [88], [96] and [70]) that endorse its expressivity, and SHACL is selected because is the latest RDF data shapes language released under the W3C recommendation umbrella. Section 5.2.2 recalls some technical characteristics presented in these three languages, since a more detailed analysis, regarding the expressiveness of the RDF constraint languages for shaping specific constraints, is carried out in section 6.5.2, where the dataset description for the RDF resources required in this thesis is available.

5.2.2 What are the technical characteristics of the RDF constraint languages selected?

The technical characteristics of the RDF constraint languages selected for shaping the OSLC resources, namely ReSh, ShEx and SHACL are presented below:

1. **Technical support**: The three languages are supported by stakeholders, and tools have been created for facilitating the creation of specifications. ReSh is endorsed by OSLC[9] and OSLC working group at OASIS[10], which its foundational sponsor is IBM[11]. The specifications created under the OSLC umbrella are supposed to be free, and software tools, to help in the implementation, extension, and testing for OSLC integrations have been released. These tools are called Eclipse Lyo[12] and OSLC4NET[13]. ReSh is a W3C recommendation since 2014. ShEx, according to [89], is not a W3C recommendation, but *it has been a major influence on the ongoing work of the W3C RDF shapes working group.* W3C experts, as well as individual researchers (that will be called others) have been working in this specification. The online tools created for working with ShEx are RDFShape[14] and the fancy ShExDemo[15]. SHACL is a new specification supported by W3C that is still a working draft specification, but available to be used. The stakeholders are SHACL W3C working group and TopQuadrant. TopQuadrant has updated its tool TopBraid Composer 5.1[16] with some of the characteristics presented in SHACL, and even though this tool lacks some of the features required for creating SHACL specification (due to ongoing development process of the tool) it is a good starting point for the verification of SHACL specifications. TopBraid Composer is not a free tool, but a there is an open version of the TopBraid Composer, which can be used for free. The

[8] https://www.w3.org/2012/12/rdf-val/
[9] http://open-services.net/
[10] http://www.oasis-oslc.org/members
[11] http://www-01.ibm.com/software/
[12] http://www.eclipse.org/lyo/
[13] http://oslc4net.codeplex.com/
[14] http://rdfshape.herokuapp.com/
[15] https://www.w3.org/2013/ShEx/FancyShExDemo
[16] http://www.topquadrant.com/technology/shacl/tutorial/

details mentioned before about the three RDF constraint languages are summarized in Table 8.

Table 8: Stakeholders and tools that support the three selected RDF constraint languages.

Language	**ReSh**	**ShEx**	**SHACL**
Stakeholder(s)	OASIS OSLC Others	W3C experts Others	W3C experts TopQuadrant
Tools	Eclipse Lyo OSLC4Net	Fancy ShExDemo RDFShape	TopBraid Composer version 5.1
W3C recommendation	Yes	No	Working draft

2. **Technical details**: The RDF constraint languages have specific characteristics. ReSh shapes can be serialized in three different forms: TURTLE syntax, XML/RDF, and JSON. The last working version is the ReSh 2.0, submitted to W3C in 2014. The vocabulary used by ReSh is the one specified for OSLC and its namespace URI is `http://open-services.net/ns/core#`, and commonly referred using the prefix *oslc*. For shaping resources using OSLC ReSh, RDF representations of the resources have to be made. ShEx shapes can be serialized in two different ways: SHEXc and JSON. There is just one working version (for practicality will be called 1.0, released in 2013 and ShEx it was not created a specific vocabulary to work with this constraint language. SHACL can be serialized in different ways: TURTLE syntax, XML/RDF and JSON. There are other serializations provided by the TobBraid Composer: XML/RDF-abbrev and N-triple. SHACL has defined its vocabulary which namespace is `http://www.w3.org/ns/shacl#` and is commonly referred using the prefix *sh*. Constraints can be defined using RDF triples and SPARQL constructs. The technical characteristics mentioned are summarized in Table 9.

Table 9: Technical characteristics of the RDF constraint languages.

Language	**ReSh**	**ShEx**	**SHACL**
Last version	OSLC ReSh 2.0 (2014)	Version 1.0 (2013)	Working draft (August 2016)
Syntax	Turtle RDF/XML JSON	SHEXc JSON	Turtle RDF/XML JSON RDF/XML-abbrev N-triple
Constraint specification	RDF triples	RDF triples	RDF triples SPARQL
Vocabulary	ReSh vocabulary	There is not specific vocabulary created for ShEx	SHACL vocabulary
namespace URI	`http://open-services.net/ns/core#`	No namespace	`http://www.w3.org/ns/shacl#`

6 Solution

This chapter presents a solution that targets the creation of an OSLC domain targeting ISO 26262, with the focus on software unit design and implementation. The methods used for proposing this solution were defined in Chapter 5 and the solution is used for generating a particular resource in chapter 7. This Chapter is structured in the following way: Subsection 6.1 presents the steps taken for creating the metamodel structure, Subsection 6.2 presents the steps taken for enriching the metamodel with constraints, Subsection 6.3 presents the steps for adding context to the domain, Subsection 6.4 presents the definition of the OSLC domain specification, Subsection 6.5 presents the definition of the shapes for the resources of the domain specification, and Subsection 6.6 presents a discussion related to the solution.

6.1 Creating a metamodel structure

For creating the metamodel structure, UML profile (explained in section 2.6.4) is used. To identify some of the elements required for the UML profile, regulatory requirements from the standard ISO 26262 are taking into account. Standard requirements were explained in Section 2.3.3. The standard gives some degree of freedom, so the aspects of the metamodel that are not modeled taking into account the standard, are taken from Scania practices. Scania practices were found in documents like those mention in Section 2.2. This section is structured in the following way: Subsection 6.1.1 presents the modeling elements of the metamodel that are taken from the standard, and Subsection 6.1.2 presents the modeling elements of the metamodel that are taken from Scania practices.

6.1.1 Modelling regulatory requirements

As mentioned in Section 2.3, ISO 26262 provides appropriate requirements and process to address risks derived from systematic failures. The requirements of the standard presented in part 6 - clause 8 (explained in Section 2.3.3) should be applied to develop a software unit in compliance with ISO 26262. The information is summarized and analyzed step by step (the following five items), so the stereotypes required for building the profile can be found.

1. *Two are the work products of this phase that corresponds to the left-hand side of the V-model: software unit design specification and software unit implementation. The software units have to be implemented as specified.* The stereotypes resulting from this requirement are:

   - software unit design specification
   - software unit implementation

   The metaclass to extend is OSLC resource, as explained in Section 5.1. The two stereotypes extend this metaclass to represent the two work products mentioned (see Figure 18). To define them as OSLC resources is adequate since these work products are characterized by many attributes, as we will see in the rest of the section.

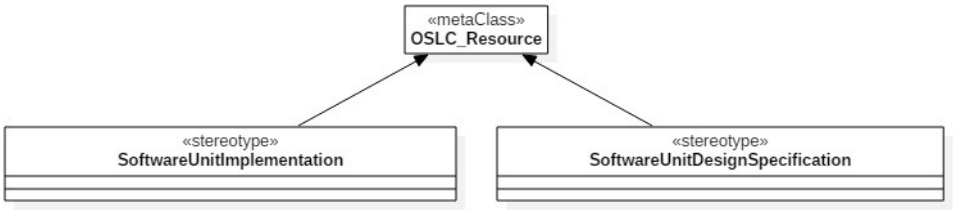


Figure 18: Representation of SoftwareUnitDesignSpecification and SoftwareUnitImplementation.

   Since the software units have to be implemented as specified, a relationship between the two classes is created. Applying the profile and adding the connection, the diagram resulting is the one in Figure 19.

Figure 19: Classes SoftwareUnitDesignSpecification and SoftwareUnitImplementation.

2. *The detailed design will be implemented as a model or directly as source code, and the requirements shall be complied with if the software unit is safety-related. The implementation include the generation of source code.* Analysing this requirement, it is found that information of how the units are implemented (model or code) must be provided, as well as the programming language. This requirement also prescribes the need to define whether the software units are safety-related or not. As we describe in Section 2.3.2, for defining the safety measures that has to be applied to an item, the item requires an ASIL. So, ASIL is the indicator if a unit is safety related (when the values A, B, C or D are assigned) or not (when the value QM is assigned). The stereotypes resulting from this requirement are:

   - ASIL
   - Implementation Type
   - Programming Language

   These stereotypes are added to the diagram as the meta-attributes *asil*, *implementationType* and *programmingLanguage* in the following way: the stereotype asil is added in both classes, since the standard prescribes that ASIL values propagate through the items of the lifecycle (ASIL decomposition is not considered in this analysis); the stereotype implementaitonType is added to the class softwareUnitDesignImplementation, taking into account that the decision of what kind of implementation would be carried out is tipically taken during design phases; and the stereotype programmingLanguage is added in the class softwareUnitImplementation, because the programming language is part of the implementation. When talking about cardinality (the amount of values that a meta-attribute is allowed to have), the analysis is the following: every software unit has assigned just one ASIL value and it is mandatory, so the cardinality is [1..1]. In the same way, the implementation type and the programming language used are just one in every software unit, and this information is also mandatory to have. So, their cardinality is also [1..1]. The result of this analysis is depicted in Figure 20.

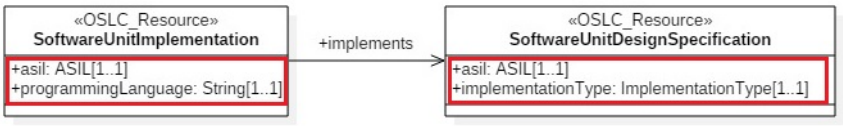


Figure 20: ASIL, Implementation Type, and Programming Language.

   It is possible to define a list of programming languages that can be used in the implementation of the software units, but this list can be as long as the number of programming languages that exist. For this reason, a *String* was defined as the data type of this meta-attribute, leaving the opportunity to write one or other language. Moreover, the allowed values of the meta-attributes asil and implementationType are predefined, and this is the reason why the value types of these meta-attributes were defined as ASIL and ImplementationType respectively. This value types are designed as enumerations in the profile (see Figure 21).

3. *The software unit design shall be described using the notations listed in Table 2 (see Section 2.3.3).* When analyzing this requirement, it is found that information about the type of notation used to describe the design must be available. The notations listed in the Table 2 are *"Natural Language, Informal notations, Semi-formal notations and Formal Notations"*. The standard also says that other notations or a combination of the ones listed in the table can be used. For this reason, a new element in this list is proposed: *"Tailored Notations"*. Moreover, the notations in the table are accompanied with a degree of recommendation, where

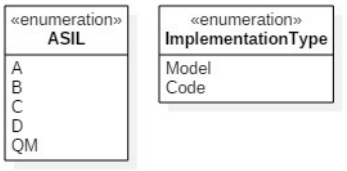


Figure 21: Enumerations ASIL and ImplementationType.

the recommendation marked as "++" (meaning *"highly recommended"*) must be preferred. If a notation with this recommendation level is not selected, a rationale must be provided. The previous analysis results in the definition of the following stereotypes:

- Design Notation Type
- Design Notation Rationale

The software unit design can be described with several notations, and at least one must be available. So, the stereotype Design Notation Type is added to the profile as the meta-attribute *designNotationType* with cardinality [1..*]. The stereotype Design Notation Rationale, which is a description of the decisions taken in selecting the design notations, and it is not mandatory when the highly recommended notations are selected, is presented a the meta-attribute designNotationRationale with cardinality [0..1]. The class is modified in the way presented Figure 22.

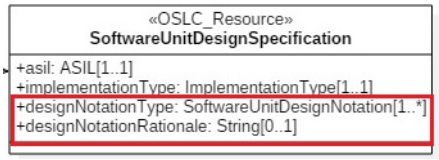


Figure 22: Design Notation Type and Design Notation Rationale.

The Design Notation Rationale is a description (in natural language) of the decisions for using the selected notations. For this reason, the data type is *String*. On the other hand, the allowed values of the meta-attribute softwareUnitDesignNotation are listed in Table 2. So, an enumeration is derived from this requirement (see Figure 23).

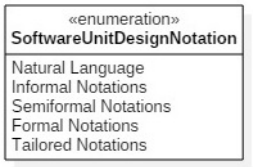


Figure 23: Enumeration softwareUnitDesignNotation.

4. *The specification of the software units shall describe the functional behavior and the internal design required for their implementation.* When analyzing this requirement, it is found that information about the functional behavior shall be available. The functional behavior represents what the software unit does. A description is also required for the understanding of the unit, so this stereotype is added. Other element suggested in this requirement is the detailed description of the internal design. However, the standard does not explicitly address the internal design (the standard just said that *"shall be described to the level of detail necessary for their implementation"*). For this reason, the internal design required for the implementation of the software unit will be addressed in the section 6.1.2, where the Scania practices will be reviewed. From this requirement, the stereotype resulting is:

   - Functional Behavior

- Description

Functional Behavior and the *Description* are written in natural language, and one value for each must be available for making these values meaningful in a unit design. So, the stereotypes are defined in the profile as two meta-attributes with data type String and cardinality [1..1]. These stereotypes are added to the class SoftwareUnitDesignSpecification as is depicted in Figure 24.

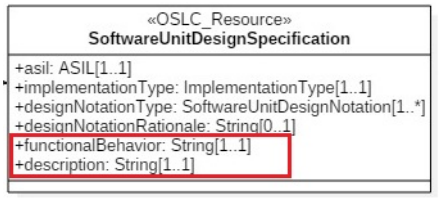


Figure 24: Representation of Functional Behavior and Description.

5. *The design principles listed in Table 3 should be applied to the design and the implementation of software units, to achieve required design properties (More information about the design properties in section 2.3.3).* This requirement is analyzed in the same way that the design notations since it corresponds to a list of properties that have to be selected according to a degree of recommendation. So, the stereotypes resulting form this requirement are:

   - Design Principle Selected
   - Design Principle Selected Rationale

   Many design principles can be selected in a design of a software unit (selected from the Table 3), and at least one must be available. So, this stereotype is added to the profile with a data type DesginPrincipleMethod (defined as an enumeration) and cardinality [1..*]. The rationale is a description (in natural language) of the decisions taken for using the selected principles. One rationale is enough for this description, and it is not necessary when the principles correspond with the highly recommended ones according to the ASIL. So, the stereotype Design Principle Selected Rationale is added to the profile with data type String and cardinality [0..1]. The meta-attributes are added to both classes SoftwareUnitDesign-Specification and SoftwareUnitImplementation, since the properties have to be reached in both stages of the lifecycle. The resulting diagram is depicted in Figure 25.

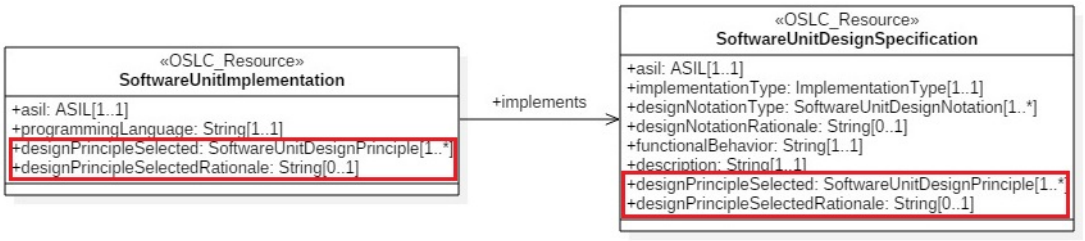


Figure 25: Representation of Design Principle Selected and Design Principle Selected Rationale.

The enumeration SoftwareUnitDesignPrinciple have the values taken from Table 3, and it is depicted in figure 26.

6.1.2 Modelling Scania practices

Practices at Scania elucidate the detailed level at what the software units are designed and implemented. The design is made as a model in *Simulink* (a model-based design program), and the implementation of the software units is generated from this model in the programming language *C* (as explained in 2.2). This information is summarized and analyzed step by step (the following four steps), so the stereotypes required for completing the profile can be found.

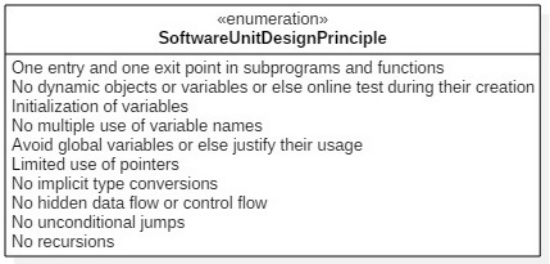


Figure 26: Enumeration SoftwareUnitDesignPrinciple.

1. *A Simulink model represents several software units. So, the generation of the code implements several software units at the time, and it shows how these units are interrelated.* When analyzing this information, a relationship between software units is found where one software unit can be related to one or more software units. In Scania, moreover, several software units are implemented together. Therefore, the relationship *implements* can be detailed with the cardinality [1..*]. The names of the classes softwareUnitDesignSpecification and SoftwareUnitImplementation can be enriched with the names found in Scania for those work products. The first one is called Simulink Model and the second one is called Application Software. The result of this analysis is depicted in Figure 27

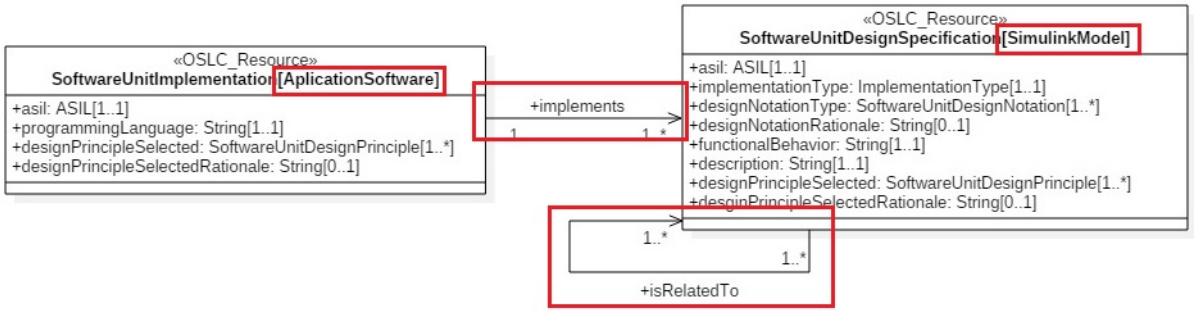


Figure 27: Relationships implements and isRelatedTo.

2. According to ISO 26262, a software unit is the smallest software representation that can have independent testing. In Scania is found that *software units that have standalone testing are composed of functions, and those functions consist of subfunctions.* This characteristic defines the stereotype *Software Unit Function*, which is defined as an OSLC resource with the meta-attribute *description*. One description is enough for every software unit function and is written in natural language. So, the cardinality of the meta-attribute description is [1..1], and the data type is String. Two relationships are also resulting from the definition of the software unit functions: a relationship called *containFunction*, which connects the class SoftwareUnitDesignSpecification, and the relationship *containSubFunction*, which is a self-reflective relationship in the class softwareUnitFunction. The first relationship has cardinality [1..*] since one software unit can have at least one, but also several software functions. The second relationship has cardinality [0..*] considering that a function can have several subfunctions, but also any subfunction. The result of this analysis is depicted in Figure 28.

3. *The software unit and the software unit functions have inputs and outputs. This inputs and outputs are treated as variables. In requirements specifications, the inputs and outputs variables of the software units are called input and output ports respectively. In the software unit functions, they are also defined as inputs and outputs variables and are called input argument and return value respectively. All variable types have the same kind of information: description and data type.* The analysis of these characteristics results in an OSLC resource called *Variable*. This resource contains the meta-attributes *description*, a string value for

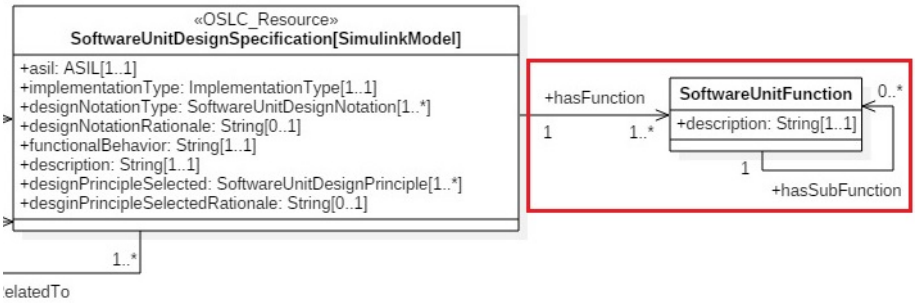


Figure 28: Software unit function class, relationships hasFunction and hasSubFunction.

allowing the definition of the variable, and *data type*, a value that can have one of the following predetermined values: string, integer, boolean, double, float, dateTime and decimal (built-in data types of XMLschema[17]]. Both, the softwareDesignUnitSpecification class and the software unitFunctionClass have defined relationships for connecting the class variable in the way depicted in Figure 29.

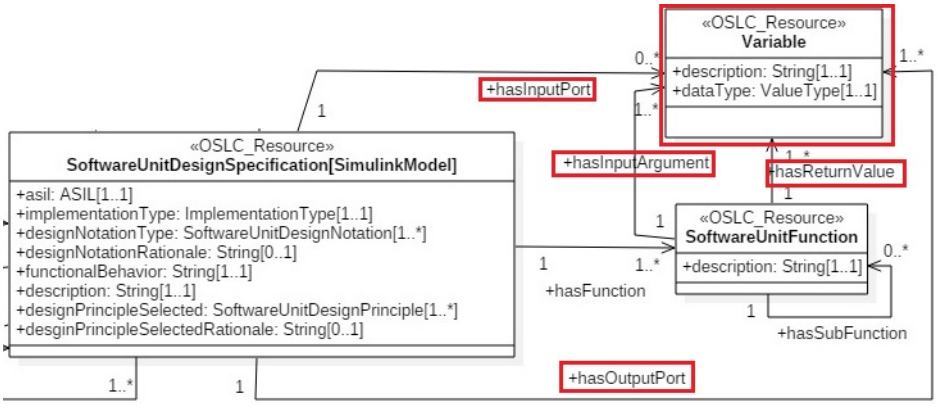


Figure 29: Class variable and the related relationships.

4. *In the document specification AE201, detailed information of the variables is written. One element found in this document is that the variables are expressed with units (they can be %, volt, state and so on). Another element found is that there are constraints on the variables. A constraint for a variable can have one of the types: required range or required accuracy. Additionally, configuration parameters are defined. Every configuration parameter has a name, a description, and several parameter values. The parameter values are accompanied by a value and a description.* The analysis of the previous characteristics results in the definition of an OSLC resource for defining the variable's constraints. The class *Constraint* has a *description* and a *constraintType* (required range or required accuracy). To relate the OSLC resource *Variable*, with the OSLC resource *Constraint*, a relationship called *isConstrainedBy* is defined. One variable can have several constraints. Two more OSLC resources result from this analysis: *ConfigurationParameter* (with meta-attributes name and description) and *ParameterValue* (with meta-attributes value and description). The class *ConfigurationParameter* is related directly with the class *SoftwareUnitDesignSpecification* (through the relationship *hasConfigurationParameter*), and the class *ParameterValue* is related with the class *ConfigurationParameter* (through the relationship *hasParameterValue*). The class *Variable* is enriched with the meta-attribute *unit*, which has value type String. Figure 30 presents the new elements added to the profile.

   In addition, the enumeration *ConstraintType*(see Figure 31) is required.

[17]http://www.w3schools.com/xml/schema_simple.asp

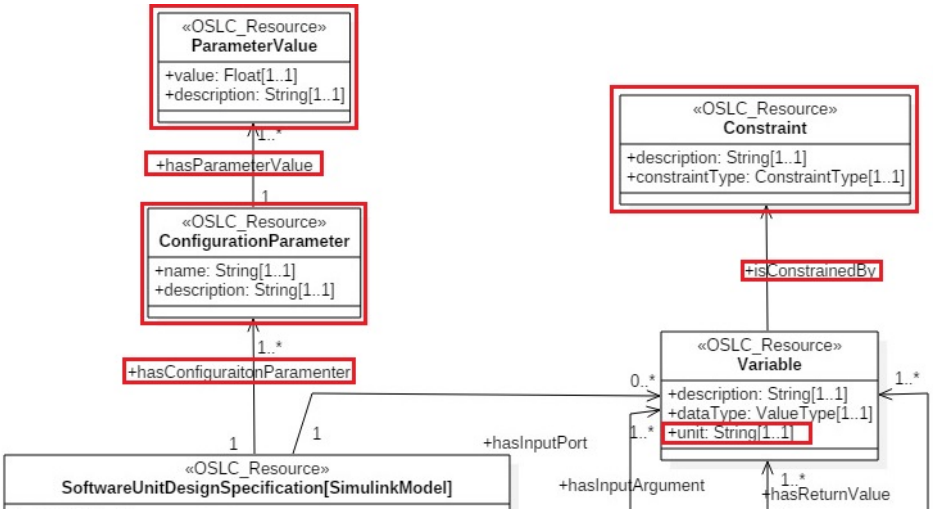


Figure 30: Definition of constraints and parameters.

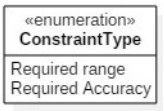


Figure 31: Enumeration ConstraintType.

6.2 Attaching constraints to the metamodel

So far, the metamodel was structured with some constraints, for example, the cardinality of the meta-attributes and relationships, as well as the allowed values (listed in enumerations) and the data types. However, other kinds of constraints should be attached to the metamodel, so proper validation of RDF data can be carried out. Following, information from the standard ISO 26262 that has to be included with the metamodel, as well as structural constraints resulting from the metamodel itself, are presented .

1. The standard says that *"ASIL values are propagated throughout the items lifecycle"*. In the metamodel created, two items of the lifecycle are modeled: *SoftwareUnitDesignSpecification* and *SoftwareUnitImplementation*. *SoftwareUnitDesignSpecification* inherits ASIL values from the architecture and the safety requirements (not modeled in the context of this thesis). This values must propagate to the *SoftwareUnitImplementation*. In the metamodel, it was decided that several SoftwareUnitDesignSpecifications instances are implemented together (given by the relationship [1..*] defined). So, a problem with the ASIL values propagation appear, since there are several values coming from several instances of the SoftwareUnitDesignSpecification, but just one can be stored in the attribute asil of the SoftwareUnitImplementation. To solve the ASIL inheritance problem, the highest ASIL of the instances of SoftwareUnitDesignSpecification that are implemented together is selected and stored in the ASIL attribute of the SoftwareUnitImplementation. The selection of the highest ASIL is supported by the standard, since this condition is prescribed there. ASIL decomposition is not evaluated in this constraint. The constraint is added to the metamodel as a note and related with the class softwareUnitImplementation (see Figure 32).

2. *The software units are implemented as specified.* A bidirectional relationship between the software unit design specification and the software unit implementation can provide a better traceability between these two work products. This constraint does not require a note, since can be defined with a structural element provided by the UML language: the two relationships *implements* and *isImplemented* (see Figure 33).

3. *Notations for software unit must be selected according to the ASIL and the recommendation*

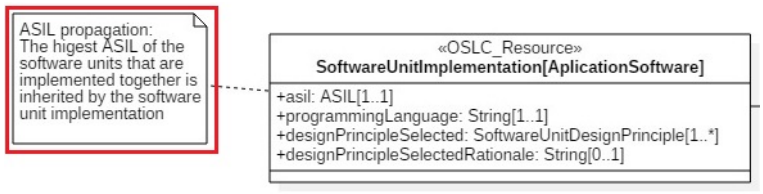


Figure 32: Constraint ASIL propagation.

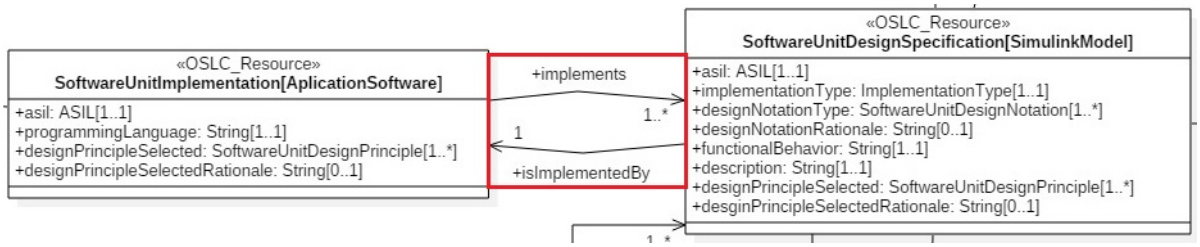


Figure 33: Bidirectional constraint implements and isImplementedBy.

level (see Table 2). If these suggestions are not followed, a rationale must be provided. This constraint represents a conditional property, that affects several meta-attributes in the meta-model. The constraint should evaluate first the ASIL assigned to the software unit design specification. With this ASIL, provide a list of design notations with the different recommendations levels (++, + and o). If design notations with recommendations levels different to "++" are selected, the meta-attribute *designNotationRationale* becomes mandatory (which means that its cardinality change from [0..1] to [1..1]). The constraint is added as a note to the OSLC resource softwareUnitDesignSpecficiation (see Figure 34).

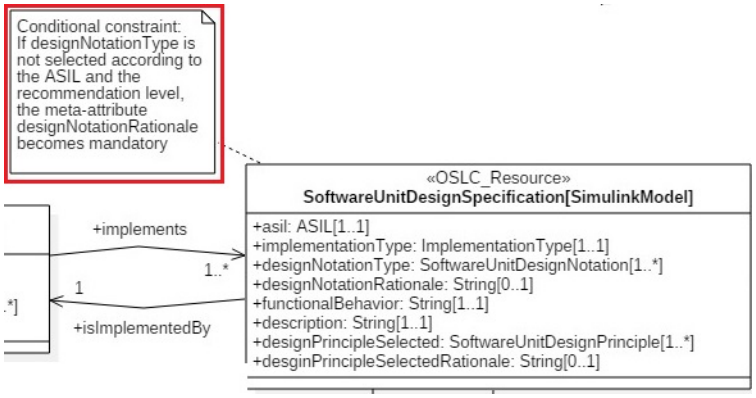


Figure 34: Conditional constraint for design notations.

An enumeration with the recommendation levels is now required for implementing the previous constraint. The values in Table 2 are presented in the following way: *Highly Recommended* corresponds with the value *++*, *Recommended* corresponds with *+* and *No recommendation*, corresponds with *o*. The enumeration is depicted in Figure 35.

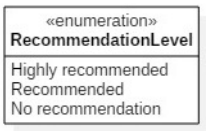


Figure 35: Enumeration RecommenationLevel.

4. *Design principles have to be also selected according to the ASIL and the recommendation level.* This constraint represents also a conditional property, with similar characteristics like the one applied to the design notations in the previous item. This constraint is added to the OSLC resources: softwareUnitDesignSpecification and softwareUnitImplementation, since these two work products require to follow the design principles. Figure 36 presents the addition of the constraint in the metamodel.

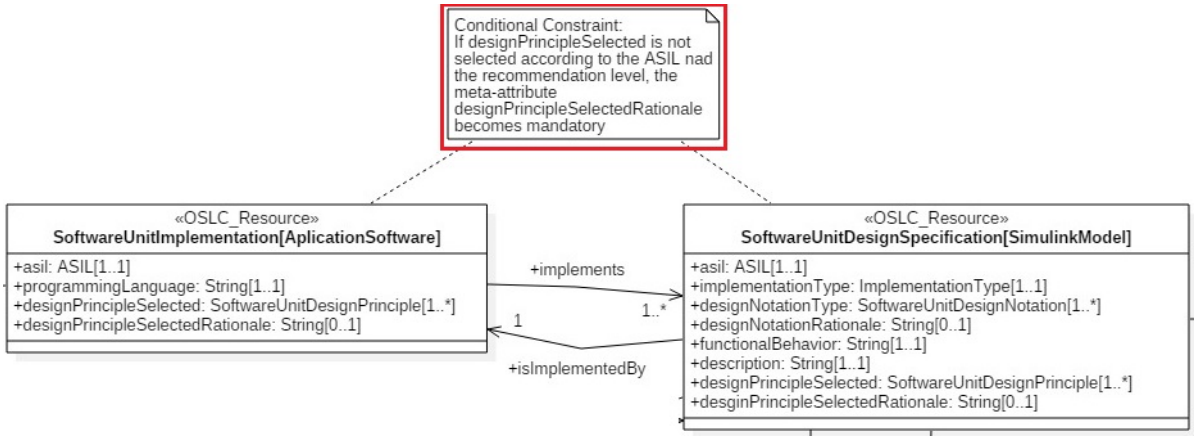


Figure 36: Conditional constraint for design principles selected.

5. *In the metamodel, the SoftwareUnitDesignSpecification has two relationships that go to the OSLC resource called Variable. These relationships are named: hasInputPort and hasOutputPort. In the Scania practices were found that and input port can not be and output port of the same software unit, so these two relationships must be disjoint.* A note with the disjoint constraint is added to the relationships mentioned before (see Figure 37).

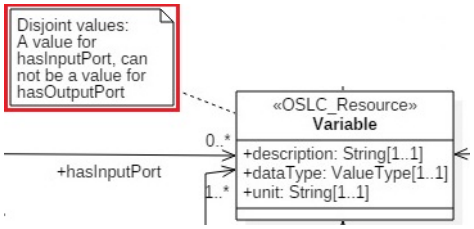


Figure 37: Disjoint constraint for input and output ports.

6. *The class SoftwareUnitFunction as the class SoftwareUnitDesignSpecification are related to variables twice. These relationships are called: hasInputArgument and hasReturnValue. An input argument can not also be a return value.* One note more, with a disjoint constraint, is added to the OSLC resource variable (see Figure 38).

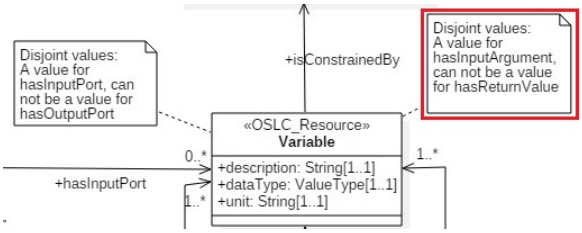


Figure 38: Disjoint constraint for input arguments and return values.

6.3 Adding context to the domain

As studied in Section 2.6.3, domains have an internal structure, but they are represented by the UML package symbols. These packages are linked to other domains with dotted lines called dependencies. Every domain also has a name and mission. This section presents the definition of this elements in the following way: Subsection 6.3.1 presents the definition of the domain name, mission and the arrangement of the internal structure, and Subsection 6.3.2 presents the relationship of the domain modeled with other domains.

6.3.1 Domain name, mission and internal structure

A knowledge domain should have a name and a clear mission. These two components provide context to the domain. From Section 5.1.1 it is recalled that one part required in the modeling is the standard ISO 26262 part 6, item 8: Software unit design and implementation. A software unit design belongs to the architectural management. From 5.1.2 is recalled that the profile created is an extension of OSLC resource domain. Taking these premises into account, the domain name should include words like OSLC, ISO 26262 and Architectural Management (AM). Since not all the Architectural Management is included, the specific parts that are involved in the domain must be specified. So, the name selected for the domain, the domain mission, as well as the elements included in the domain are presented in Figure 39. Elements in the figure were colored so they can be identified easier: elements colored in green are the OSLC resources, elements colored in pink are constraints, and elements colored in yellow are enumerations.

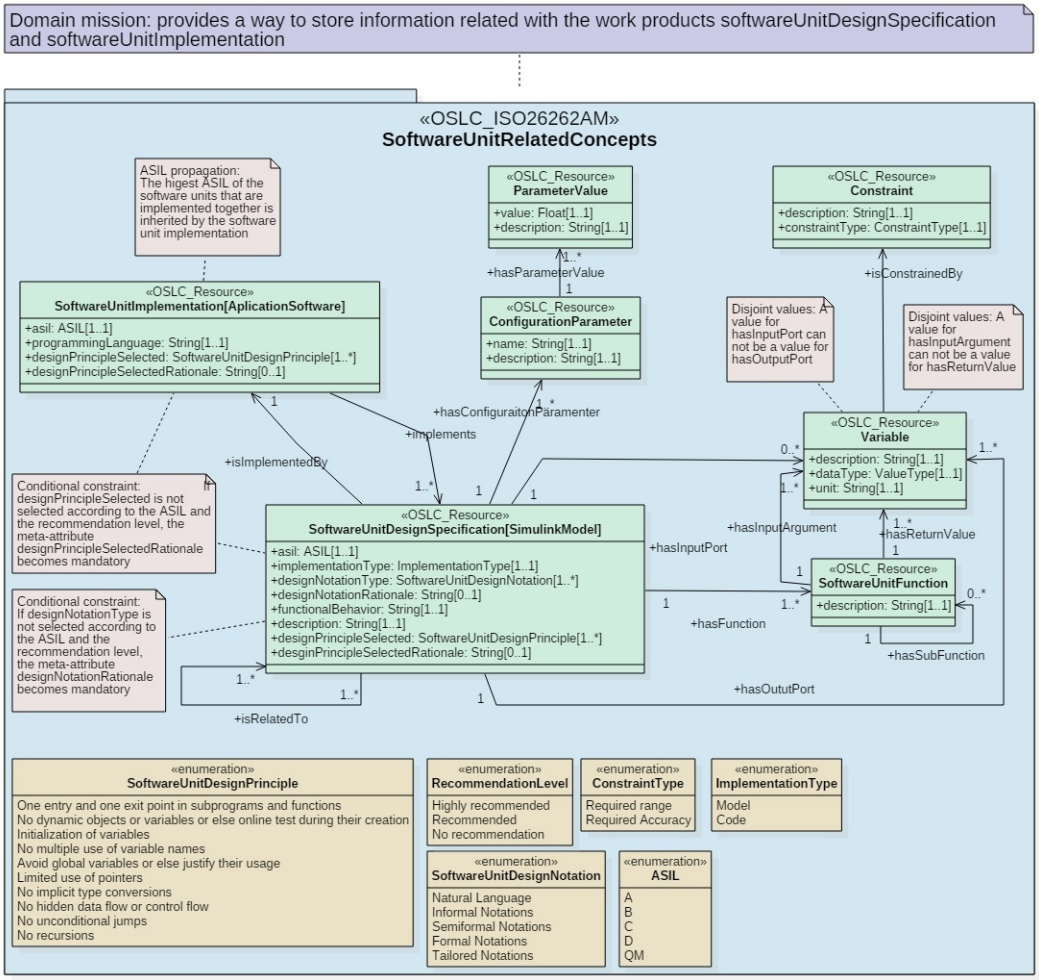


Figure 39: Profile diagram: SoftwareUnitRelatedConcepts.

6.3.2 Relationships with other domains

Each domain provides services and makes use of other domains. Since this knowledge domain is an element of the ISO 26262 lifecycle, the standard is taken again to search traces between the domain created and others. The standard parts and its analysis are presented as follows:

1. *"The first objective of this sub-phase is to specify the software units in accordance with software architectural design and the associated software safety requirements"*. This requirement suggests a relationship between the software architectural design related concepts, the software safety requirements related concepts and the software unit related concepts. As its name suggest, software architectural design belongs also to the AM domain, but the software safety requirements related concepts belong to a domain called Requirements Management.
2. *"The third objective of this sub-phase is the static verification of the design of the software units and their implementation"*. This requirements suggests a relationship between the verification related concepts and the software unit related concepts. According to [81], verification related concepts belongs to a domain called Quality Management.

As was mention in Section 2.6.2, a bridge exists between the domains, where a domain makes assumptions, and the other domain takes those assumptions as requirements. For representing the bridge, a dotted dependency is drawn. The result of the addition of domains and bridges is the domain chart presented in Figure 40.

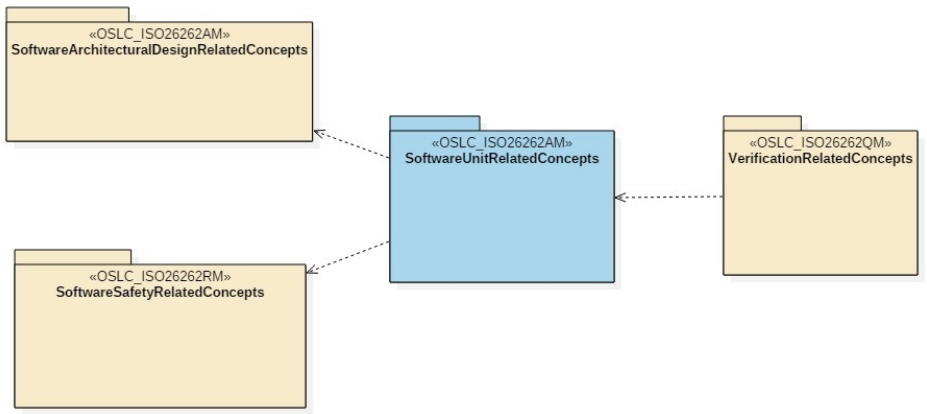


Figure 40: Domain chart.

6.4 Defining OSLC domain specification

For defining the OSLC resources, an OSLC domain specification must be provided. This domain specification provides the vocabulary terms required for the creation of OSLC resources. It is built taken into account the elements provided by the domain chart created in section 6.1 and the definition of OSLC property constraints explained in Section 2.9.4. One of the first elements required for the definition of a domain specification is to determine the XML namespace (which is composed of a URI and a prefix). The idea of this thesis is that the domains specified here become candidate for OSLC specifications, so the XML namespace proposed follows the standard defined by OSLC community. For this, a proposal XML namespace is:

- URI: `http://open-services.net/ns/iso26262am#`
- Prefix: oslc_iso26262am

Once the XML namespace is determined, the OSLC property constraints are defined for each OSLC resource specified in the knowledge domain (see Figure 40). In Section 5.1.3 it was specified the vocabularies that would be part of the definition of the OSLC resources. The vocabularies are:

- Dublin Core Metadata Initiative Terms, which has URI `http://purl.org/dc/terms/` and prefix dcterms.
- OSLC core, which has URI `http://open-services.net/ns/core#` and prefix oslc.
- XML Schema which has URI `http://www.w3.org/2001/XMLSchema` and prefix xsd.

Seven OSLC resources are defined, and they are presented as follows:

1. *OSLC resource SoftwareUnitDesignSpecification*: This OSLC resource is composed of eight meta-attributes and five relationships. Table 10, presents the properties defined for this resource.
2. *OSLC resource SoftwareUnitImplementation*: This OSLC resource is composed by four meta-attributes and one relationship. Table 11 presents the definition of the properties for this resource.
3. *OSLC resource Variable*: This OSLC resource is composed of three meta-attributes and one relationship. Table 14 presents the definition of the properties for this resource.
4. *OSLC resource Constraint*: This OSLC resource is composed of two meta-attributes. It does not have relationships. Table 12 presents the definitions of the properties for this resource.
5. *OSLC resource ParameterValue*: This OSLC resource is composed by two meta-attributes. Table **??** presents the definition of the properties for this resource.
6. *OSLC resource ConfigurationParameter*: This OSLC resource is composed by two meta-attributes and one relationship. Table 15 presents the definition of the properties for this resource.
7. *OSLC resource SoftwareUnitFunction*: This OSLC resource is composed of one meta-attribute and three relationships. The meta-attribute is called description, and its property definitions is taken from dcterms. The two relationships define two new terms in the domain vocabulary. Table 16 presents the definition of the properties for this resource.

Table 10: OSLC definition for the resource SoftwareUnitDesignSpecification.

Prefixed Name	**Occurs**	**Value Types**	**Rep.**	**Range**	**Description**
Properties: meta-attributes					
oslc_iso26262am: asil	Exactly one	Local resource	Inline	Any	Defines the level to specify the items necessary requirements of ISO 26262.
oslc_iso26262am: implementation-Type	Exactly one	Local resource	Inline	Any	Describes the type of implementation type used to create the software.
oslc_iso26262am: designNotation-Type	One or many	Local resource	Inline	Any	Describes the type of design notation used to specify the unit.
oslc_iso26262am: designNotation-Rationale	Zero or one	String	n/a	n/a	Describes the rationale behind the design notation method selected.
oslc_iso26262am: functionalBehavior	Exactly one	String	n/a	n/a	Describes the functional behaviour of the software unit.
dcterms: description	Exactly one	String	n/a	n/a	Descriptive text of the software unit that is specified.
oslc_iso26262am: designPrincipleSelected	One or many	Local resource	Inline	Any	Describes the design principle selected.
oslc_iso26262am: designPrincipleSelectedRationale	Zero or one	String	n/a	n/a	Describes the rationale behind the design principle method selected.
Properties: relationships					
oslc_iso26262am: isRelatedTo	One or many	Resource	Reference	**oslc_iso26262am:** softwareUnitdesignSpecification	A relationship that describes the connection between a software unit design specification and other software unit design specifications.
oslc_iso26262am: hasFunction	One or many	Resource	Reference	**oslc_iso26262am:** softwareUnit-Function	A relationship that describes the connection between a software unit design specification and its functions.
oslc_iso26262am: hasInputPort	One or many	Resource	Reference	**oslc_iso26262am:** Variable	A relationship that describes the connection between a software unit design specification and its input ports.
oslc_iso26262am: hasOutputPort	One or many	Resource	Reference	**oslc_iso26262am:** Variable	A relationship that describes the connection between a software unit design specification and its output ports.
oslc_iso26262am: hasConfigurationParameter	One or many	Resource	Reference	**oslc_iso26262am:** ConfigurationParameter	A relationship that describes the connection between a software unit design specification and its configuration parameters.
oslc_iso26262am: isImplementedBy	Exactly one	Resource	Reference	**oslc_iso26262am:** SoftwareUnitImplementation	A relationship that describes the connection between a software unit design specification and the software unit implementation.
Proposed IRI: `http://open-services.net/ns/iso26262am#softwareUnitDesignSpecification`					

Table 11: OSLC definition for the resource SoftwareUnitImplementation.

Prefixed Name	**Occurs**	**Value Types**	**Rep.**	**Range**	**Description**
Properties: meta-attributes					
oslc_iso26262am: asil	Exactly one	Local resource	Inline	Any	Defines the level to specify the items necessary requirements of ISO 26262. Allowed values are: A, B, C,D, or QM.
oslc_iso26262am: programmingLanguage	Exactly one	String	n/a	n/a	Defines the programming language used to generate source code of the software units.
oslc_iso26262am: designPrincipleSelected	One or many	Local resource	Inline	Any	Describes the design principle selected. Allowed values in Table 3
oslc_iso26262am: designPrincipleSelectedRationale	Zero or one	String	n/a	n/a	Describes the rationale behind the design principle method selected.
Properties: relationships					
oslc_iso26262am: implements	One or many	Resource	Reference	**oslc_iso26262am:** softwareUnitdesignSpecification	A relationship that describes the connection between a software unit implementation and its software unit design specifications.
Proposed IRI: `http://open-services.net/ns/iso26262am#softwareUnitDesignImplementation`					

Table 12: OSLC definition for the resource Constraint.

Prefixed Name	**Occurs**	**Value Types**	**Rep.**	**Range**	**Description**
Properties: meta-attributes					
dcterms: description	Exactly one	String	n/a	n/a	Descriptive text of the constraint that is specified.
oslc_iso26262am: constraintType	Exactly one	Local resource	Inline	Any	Describes the type of the constraint. Allowed values are: Required range or Required accuracy.
Properties: relationships					
No relationships					
Proposed IRI: `http://open-services.net/ns/iso26262am#Constraint`					

Table 13: OSLC definition for the resource ParameterValue.

Prefixed Name	**Occurs**	**Value Types**	**Rep.**	**Range**	**Description**
Properties: meta-attributes					
oslc: name	Exactly one	String	n/a	n/a	Represent the name of the ConfigurationParameter.
dcterms: description	Exactly one	String	n/a	n/a	Descriptive text of the parameter value that is specified.
Properties: relationships					
No relationships					
Proposed IRI: `http://open-services.net/ns/iso26262am#ParameterValue`					

Table 14: OSLC definition for the resource Variable.

Prefixed Name	Occurs	Value Types	Rep.	Range	Description
Properties: meta-attributes					
dcterms: description	Exactly one	String	n/a	n/a	Descriptive text of the variable that is specified.
oslc_iso26262am: dataType	Exactly one	Resource	Reference	**xsd:** valueType	Describes the data type of the variable. Allowed values are: boolean, dateTime, decimal, double, float,integer, string.
oslc_iso26262am: unit	Exactly one	String	n/a	n/a	Describes the unit in which the variable is expressed.
Properties: relationships					
oslc_iso26262am: isConstrainedBy	Zero or many	Resource	Reference	**oslc_iso26262am:** Constraint	A relationship that describes the connection between a variable and its constraints.
Proposed IRI: `http://open-services.net/ns/iso26262am#Variable`					

Table 15: OSLC definition for the resource ConfigurationParameter.

Prefixed Name	Occurs	Value Types	Rep.	Range	Description
Properties: meta-attributes					
oslc: name	Exactly one	String	n/a	n/a	Represent the name of the ConfigurationParameter.
dcterms: description	Exactly one	String	n/a	n/a	Descriptive text of the configuration parameter that is specified.
Properties: relationships					
oslc_iso26262am: hasParameterValue	One or many	Resource	Reference	**oslc_iso26262am:** ParameterValue	A relationship that describes the connection between a configuration paramenter and its values.
Proposed IRI: `http://open-services.net/ns/iso26262am#ConfigurationParameter`					

Table 16: OSLC definition for the resource SoftwareUnitFunction.

Prefixed Name	Occurs	Value Types	Rep.	Range	Description
Properties: meta-attributes					
dcterms: description	Exactly one	String	n/a	n/a	Descriptive text of the software unit that is specified.
Properties: relationships					
oslc_iso26262am: hasSubFunction	One or many	Resource	Reference	**oslc_iso26262am:** softwareUnitFunction	A relationship that describes the connection between a software unit function and its subfunctions.
oslc_iso26262am: hasInputArgument	One or many	Resource	Reference	**oslc_iso26262am:** Variable	A relationship that describes the connection between a software unit function and its input arguments.
oslc_iso26262am: hasReturnValue	One or many	Resource	Reference	**oslc_iso26262am:** Variable	A relationship that describes the connection between a software unit function and its return values.
Proposed IRI: `http://open-services.net/ns/iso26262am#SoftwareUnitFunction`					

6.5 Shaping the RDF resources

This section is created for exploring the constraints found on the knowledge domain presented in Figure 39 and classify them according to certain criteria. Once the constraints of RDF data are listed, they are mapped to the RDF data shapes languages selected (ReSh, ShEx and SHACL), in order to find a RDF data shape language candidate for the definition of the shapes for the domain oslc_iso26262am: SoftwareUnitRelatedConcepts. This section is structured in the following way: Subsection 6.5.1 presents the definition of the requirements for a RDF constraint language; Subsection 6.5.2 presents the mapping of the requirements for a RDF constraint language to RDF data shapes languages; and Subsection 6.5.3 presents the summary of the comparative study related to constraint languages.

6.5.1 Defining the requirements for a RDF constraint language

As mention in Section 2.8.2, the W3C RDF Data Shapes working group is aiming for finding a proper validation for RDF data. One of its contributions is the creation of a database of requirements on RDF constraints validation and formulation[18]. This database is a space were data practitioners can contribute with their own RDF data constraints needs. Recently, an evaluation of RDF validation requirements presented in this database were made by Bosch et al. [88]. This last job describes in detail each requirement within the RDF validation requirements database. For this reason, it is taken into account for naming and describing the validation constraints found in the knowledge domain presented in Figure 39. The next list, which have in total ten constraints, presents the classification of the requirements for RDF constraint languages.

1. **Requirement 1 - Disjoint properties**. This requirement is defined in the following way: *"A disjoint property states that all the properties are pairwise disjoint. It means that an individual x can not be connected to an individual y by these property"*. In the domain specification for software unit related concepts, this requirement is found in the following properties:

   - The properties hasInputPort and hasOutputPort are disjoint. It is expressed as *disjoint(hasInputPort, hasOutputPort)*.
   - The properties hasInputArgument and hasReturnValue are disjoint. It is expressed as *disjoint(hasInputArgument, hasReturnValue)*.

2. **Requirement 2 - Minimum qualified cardinality restrictions**. This requirement is defined in the following way: *"it contains all those individuals that are connected by a property to at least n different individuals of a particular class or data range"*. In the domain specification for software unit related concepts, this requirement is found in all the properties, and it corresponds with the number written in the left side of the pair [min..max] which accompany all the properties in Figure 39. For example, property designNotationType in the OSLC resource SoftwareUnitDesignSpecification which is presented with cardinality [1..*], has minimum qualified cardinality 1.

3. **Requirement 3 - Maximum qualified cardinality restrictions**. This requirement is defined in the following way: *"it contains all those individuals that are connected by a property to at most n different individuals/literals that are instances of a particular class or data range"*. In the domain specification for software unit related concepts, this requirement is found in all the properties, and it corresponds with the number written in the right side of the pair [min..max] which accompany all the properties in Figure 39. For example, property designNotationType in the OSLC resource SoftwareUnitDesignSpecification which is presented with cardinality [1..*], has maximum qualified cardinality *, which means *"many"*.

4. **Requirement 4 - Exact qualified cardinality restrictions**. This requirement is defined in the following way *"An exact cardinality restriction contains all those individuals that are connected by a property to exactly n different individuals that are instances of a particular*

[18] http://lelystad.informatik.uni-mannheim.de/rdf-validation/

class or data range.". In the domain specification for software unit related concepts, this requirement is found in those properties that are described in the Figure 39 with [1..1].

5. **Requirement 5 - Cardinality shortcuts**. Requirements 2, 3 and 4 can be grouped and defined together in a requirement called *cardinality shortcuts*. This requirement is defined in the following war: *"pair of values that establish maximum and minimum cardinality"*. The list of shortcuts is the following:

   - Optional and non-repeatable [0..1]
   - Optional and repeatable [0..*]
   - Mandatory and non-repeatable [1..1]
   - Mandatory and repeatable[1..*]

6. **Requirement 6 - Inverse object property**. This requirement is defined in the following way *"properties are used bidirectionally and then accessed in the inverse direction"*. In the domain specification for software unit related concepts, this requirement is found in the pair of properties: *implements* and *isImplementedBy*

7. **Requirement 7 - Context-specific Exclusive Or of properties groups**. This requirement is defined in the following way: *"inclusive OR is a logical connective joining two or more predicates that yields the logical value true when at least one of the predicates is true"*. In the domain specification for software unit related concepts, for example, this requirement is found when there is one value for asil and a designNotationType (with a highly recommended value) OR asil and designNotationType (with a recommended value) and designNotationRationale.

8. **Requirement 8 - Allowed values**. This requirement is defined in the following way: *"it is an exhaustive enumeration of the valid values"*. Allowed values can be:

   - literals of a list of allowed literals, for example, the allowed values for the asil property are "A","B", "C", "D", "QM".
   - types literals of specific types, for example, a description has to be a string or a value has to be a float.

9. **Requirement 9 - Conditional properties**. This requirement is defined in the following way: *"if specific properties are present, then specific other properties have to be present"*. In the domain specification for software unit related concepts, for example, if the value for designNotationType do not corresponds with a highly recommended value then the designNotationRationale property is mandatory.

10. **Requirement 10 - Handle RDF collections**. This requirement is defined in the following way: *"it is a requirement that handles different types of RDF requirements in the collections, like the comparison of elements in a collection"*. In the domain specification for software unit related concepts, for example, the asil property of the software units that are implemented together, have to be compared, so the highest ASIL is inherited by the property asil of the softwareUnitImplementation. This property may permit the generation of the constraint name in Figure 39 *ASIL propagation*.

6.5.2 Mapping requirements for RDF constraints to RDF data shapes languages

In this section, a mapping of the requirements for RDF constraints to the RDF data shapes languages is presented. For this, the list of requirements presented in section 6.5.1 is recalled and developed in the languages that support it. To distinguish the language used in the mapping, a prefix, that shows the name of the language is selected, i.e. for ReSh, the prefix is **oslc_iso26262am**; for ShEx, the prefix is **shex_iso26262am**; and for SHACL, the prefix is **shacl_iso26262am**.

1. **Requirement 1 - Disjoint properties**. ReSh and ShEx do not provide a construct for making properties disjoint, but SHACL provides the construct *sh:disjoint*, which *"constraints a pair of properties so that the set of values of both properties at a given focus node must not share any nodes"* [9]. The fragment of SHACL program presented in Listing 9, shows how the construct sh:disjoint is used.

```
sh:property [
      sh:class shacl_iso26262am:Variable ;
      sh:disjoint shacl_iso26262am:hasInputPort ;
      sh:minCount 1 ;
      sh:name "has output port"^^xsd:string ;
      sh:predicate shacl_iso26262am:hasOutputPort ;
    ] ;
```

Listing 9: Disjoint properties in SHACL.

Listing 9, presents the definition of the property *hasOutputPort* (this definition is made with the construct *sh:predicate shacl_iso26262am:hasOutputPort*. For disjoining this property from the property hasInputPort, the element *sh:disjoint shacl_iso26262am:hasInputPort* is added in the definition of the property.

2. **Requirement 2 - Minimum qualified cardinality restrictions**. ReSh and ShEx do not provide a constructor for defining minimum qualified cardinality. In SHACL the construct *sh:minCount* is used for defining this requirement. When the minimum cardinality is defined in SHACL, but not the maximum, it means that the property have at least 1 instance, but also can have many. The property *isRelatedTo* is presented in Listing 10, where a minimum cardinality restriction is defined.

```
  sh:property [
      sh:class shacl_iso26262am:SoftwareUnitDesignSpecification ;
      sh:description """A relationship that describes the connection
          between a software unit design specification and other
          software unit design specifications."""^^xsd:string ;
      sh:minCount 1 ;
      sh:name "is related to"^^xsd:string ;
      sh:nodeKind sh:IRI ;
      sh:predicate shacl_iso26262am:isRelatedTo ;
    ] ;
```

Listing 10: Definition of minimum qualified cardinality restrictions in SHACL.

3. **Requirement 3 - Maximum qualified cardinality restrictions**. ReSh and ShEx do not provide a constructor for defining maximum qualified cardinality. In SHACL the construct *sh:maxCount* is used for defining this requirement. In the initial definition of the domain *softwareUnitRelatedConcepts*, the property *designNotationRationale* is created as optional. It means that the maximum cardinality is one, but it is not mandatory, so the minimum cardinality is zero. This is expressed in SHACL, using the construct *sh:maxCount*, but it does not need the construct *sh:minCount* to define that zero is the minimun. Listing 11, shows the definition of the property mentioned.

```
  sh:property [
      sh:datatype xsd:string ;
      sh:description "Describes the rationale behind the design notation
          method selected."^^xsd:string ;
      sh:maxCount 1 ;
      sh:name "design notation rationale"^^xsd:string ;
      sh:predicate shacl_iso26262am:designNotationRationale ;
    ] ;
```

Listing 11: Definition of minimum qualified cardinality restrictions in SHACL.

4. **Requirement 4 - Exact qualified cardinality restrictions**. ReSh and ShEx can express exact cardinality restrictions when it is [1..1] or [0..1] (this is cover by the requirement 5 - cardinality shortcuts) but can not define other kind of limits. In SHACL the limits of the cardinality can be defined, using the constructs *sh:minCount* and *sh:maxCount* together. The property designNotationType, is a property have maximum five allowed values, defined in the enumeration *SoftwareUnitDesignNotations*. Since it does not have meaning if there are more that five values selected for this property, it can be defined with exact qualified cardinality [1..5]. The property is presented in Listing 12, where maximum and minimum cardinality are defined.

```
sh:property [
    sh:class shacl_iso26262am:SoftwareUnitDesignNotation ;
    sh:description "Describes the type design notation method used to
        specify the unit. The value can be: Natural language,
        Informal notations, Semiformal Notations, Formal Notations or
        Tailored Notation."^^xsd:string ;
    sh:maxCount 5 ;
    sh:minCount 1 ;
    sh:name "design notation type"^^xsd:string ;
    sh:nodeKind sh:IRI ;
    sh:predicate shacl_iso26262am:designNotationType ;
  ] ;
```

Listing 12: Definition of the exact cardinality in SHACL.

5. **Requirement 5 - Cardinality shortcuts**. This implementation covers the requirements 2 and 3 defined in section 6.5.1. The requirement 4 is partially covered, when the exact cardinality is [0..1] or [1..1]. Cardinality shortcuts are present in ReSh and ShEx, but not in SHACL (but they can be expressed with the construct *sh:minCount* and *sh:maxCount* used together). In ReSh, the construct *oslc:occurs* is used to define the cardinality of the property. The shortcut available in ReSh are:

   - `http://open-services.net/ns/core#Exactly-one`
   - `http://open-services.net/ns/core#Zero-or-one`
   - `http://open-services.net/ns/core#Zero-or-many`
   - `http://open-services.net/ns/core#One-or-many`

   The definition of the property functionalBehavior, with cardinality shortcut *Exactly-One* in ReSh is presented in Listing 13.

```
<!--Property 3: functionalBehavior-->
  <oslc:property>
     <oslc:Property>
        <oslc:name>functionalBehavior</oslc:name>
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#
            Exactly-one"/>
        <oslc:propertyDefinition rdf:resource="http://open-services.net
            /ns/iso26262#functionalBehavior"/>
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#
            string"/>
        <oslc:readOnly>true</oslc:readOnly>
     </oslc:Property>
  </oslc:property>
```

Listing 13: Definition of cardinality shortcuts in ReSh.

   In ShEx, cardinality constraints are expressed with the signs ?, +, *, {y,z} or any symbol, as it is explained in Table 4. The property functionalBehavior defined in ShEx is presented in Listing 14. The property does not have any symbol after the assignation of the data type *xsd:string*, which means that the cardinality is Exactly-one.

```
#Property: functional Behaviour
   shex_iso26262am:functionalBehaviour xsd:string ,
```

Listing 14: Definition of cardinality shortcuts in ShEx.

As explained before, cardinality shortcuts do not exists in SHACL, but cardinality can be expressed expressed separately with the constructs *sh:minCount* and *sh:maxCount* used together. The property functionalBehavior is expressed in SHACL, in Listing 15.

```
  sh:property [
      sh:datatype xsd:string ;
      sh:description """Describes the functional behaviour of the
          software unit."""^^xsd:string ;
      sh:maxCount 1 ;
      sh:minCount 1 ;
      sh:name "functional behavior"^^xsd:string ;
      sh:predicate shacl_iso26262am:functionalBehavior ;
    ] ;
```

Listing 15: Definition of cardinality shortcuts in SHACL.

6. **Requirement 6 - Inverse object property**. This requirement can be fulfilled in ShEx and SHACL directly. In ReSh the definition of two properties - relationships should partially cover the bidirectionality of them. In ShEx, the direction of the relationship is changed by prefixing the triple constraint with the symbol ^. The property *isImplementedBy* is the inverse of the property *implements* (see Figure 39) and it is expressed in ShEx in Listing 16.

```
#Shape: Software unit design Specification
<SoftwareUnitDesignSpecificationShape>{
.
. #other properties
.
 #Relationship: isImplementdBy
   ^shex_iso26262am:implements @<SoftwareUnitImplementationShape>+
}

#Shape: Software unit Implementation
<SoftwareUnitImplementationShape>{

.
. #definition of the properties
.
 #Relationship: implements
   ^shex_iso26262am:isImplementsBy @<SoftwareUnitDesignSpecification>
}
```

Listing 16: Definition of Inverse property in ShEx.

In Listing 16 the relationship *isImplementedBy* is defined as an inverse property in the shape *SoftwareUnitImplementation* and the the relationship *implements* is defined as an inverse property in the shape *SoftwareUnitDesignSpecification*. With exemplification purposes, both relationships were defined, but just one is required.

In SHACL is applied the same principle that is used in ShEx. In listing 17, the class defined is *softwareUnitDesignSpecification* and the property-relationship *implements*, is defined as an inverse property in this class, using the construct *sh:inverseProperty*.

```
shacl_iso26262am:SoftwareUnitDesignSpecification
  rdf:type rdfs:Class ;
  rdf:type sh:Shape ;
```

```
  rdfs:label "Software unit design specification"^^xsd:string ;
  sh:inverseProperty [
      sh:class shacl_iso26262am:SoftwareUnitImplementation ;
      sh:description "it is an inverse property were the relationship
          describes the connection between a software unit
          implementation and its software unit design specifications"^^
          xsd:string ;
      sh:minCount 1 ;
      sh:name "implements"^^xsd:string ;
      sh:nodeKind sh:IRI ;
      sh:predicate shacl_iso26262am:implements ;
    ] ;
```

Listing 17: Definition of Inverse property in SHACL.

Inverse properties are not possible to define in ReSh. However, some kind of bidirectionality can be expressed, defining both relationships. Listing 18 presents the definition of the properties *isImplementBy*, where the range is SoftwareUnitImplementation (this marks the direction of the relationship). Listing 19 presents the definition of the properties *implements*, where the range is SoftwareUnitDesignSpecification (this marks the direction of the relationship).

```
<!--Property: relationship "isImplementedBy"-->
  <oslc:property>
     <oslc:Property>
        <oslc:name>isImplementedBy</oslc:name>
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#
            Exaclty_one"/>
        <oslc:propertyDefinition rdf:resource="http://open-services.net
            /ns/iso26262#isImplementedBy"/>
        <oslc:valueType rdf:resource="http://open-services.net/ns/core#
            Resource "/>
        <oslc:representation rdf:resource="http://open-services.net/ns/
            core#Reference"/>
        <oslc:range rdf:resource="http://open-services.net/ns/iso26262#
            SoftwareUnitImplementation"/>
        <oslc:readOnly>true</oslc:readOnly>
     </oslc:Property>
  </oslc:property>
```

Listing 18: Definition of the property isImplementedby in ReSh.

```
<!--Property: relationship "implements"-->
  <oslc:property>
     <oslc:Property>
        <oslc:name>implements</oslc:name>
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#One
            -or-many"/>
        <oslc:propertyDefinition rdf:resource="http://open-services.net
            /ns/iso26262#implements"/>
        <oslc:valueType rdf:resource="http://open-services.net/ns/core#
            Resource "/>
        <oslc:representation rdf:resource="http://open-services.net/ns/
            core#Reference"/>
        <oslc:range rdf:resource="http://open-services.net/ns/iso26262#
            SoftwareUnitDesignSpecification"/>
        <oslc:readOnly>true</oslc:readOnly>
     </oslc:Property>
  </oslc:property>
```

Listing 19: Definition of the property implements in ReSh.

7. **Requirement 7 - Context-specific Exclusive Or of properties groups**. This property can not be implemented in ReSh. In ShEx the vertical bar is used for implementing Exclusive Or, so properties can be grouped together. Listing 20 shows the following condition: When it is selected ASIL A, the value of the property havingAsilA has to be true. When RecommendationLevel is assigned highlyRecommended, the designNotationType is restricted to natural language and informal notations. In case that the recommendationLevel assigned is recommended, the designNotationType property is restricted to the values semiformalNotations, formalNotations and TailoredNotations, and the property designNotationRationale becomes mandatory.

```
<SoftwareUnitDesignSpecification> {
   # Assigns ASIL A to the software unit
   (shex_iso26262am:havingAsilA(shex_iso26262am:true),
   # Assigns highly Recommended type to the software unit
   shex_iso26262am:recommendationLevel(shex_iso26262am:highlyRecommended
       ),
   # Restricts the selection of design notation type to those allowed
       for highly recommended
   shex_iso26262am:designNotationType(shex_iso26262am:naturalLanguage
       shex_iso26262am:informalNotations)| Operator Exclusive Or (|)
   # Assigns ASIL A to the software unit
   shex_iso26262am:havingAsilA(shex_iso26262am:true),
   # Assigns Recommended type to the software unit
   shex_iso26262am:recommendationLevel(shex_iso26262am:recommended),
   # Restricts the selection of design notation type to those allowed
       for recommended type
   shex_iso26262am:designNotationType(
       shex_iso26262am:semiformalNotations
       shex_iso26262am:formalNotations shex_iso26262am:TailoredNotations
       ),
   #Makes the property designNotationRationale mandatory
   shex_iso26262am:designNotationRationale xsd:string|
#(...Same kind of constraints for ASIL B,C and D are defined here, QM
    restricts only to highly recommended level)
}
```

Listing 20: Grouping properties using ShEx.

In SHACL exists the construct *sh:or* which provides a similar function that the OR in ShEx. Listing 21 shows the similar functionality reached in ShEx in Listing 20, but using a shape in SHACL.

```
shacl_iso26262am:AsilAHighlyRecommendedLevels
  rdf:type sh:Shape ;
  rdfs:label "Asil A Highly Recommended levels"^^xsd:string ;
  sh:constraint [
      sh:or (
          [
            sh:property [
                sh:hasValue shacl_iso26262am:A ;
                sh:minCount 1 ;
                sh:predicate shacl_iso26262am:asil ;
              ] ;
          ]
          [
            sh:property [
                sh:hasValue shacl_iso26262am:Highly_Recommended ;
                sh:minCount 1 ;
                sh:predicate shacl_iso26262am:recommendationLevel ;
              ] ;
          ]
```

```
          [
            sh:property [
                sh:in (
                    shacl_iso26262am:NaturalLanguage
                    shacl_iso26262am:InformalNotations
                  ) ;
                sh:minCount 1 ;
                sh:predicate shacl_iso26262am:designNotationType ;
              ] ;
          ]
        ) ;
    ] ;
  sh:scopeClass shacl_iso26262am:SoftwareUnitDesignSpecification ;
.
```

Listing 21: Grouping properties using SHACL.

8. **Requirement 8 - Allowed values**. Allowed values is a requirement that can be implemented in the three languages. In ReSh, it is implemented using the construct *oslc:allowedValue*. Listing 22 presents the definition of the property implementationType, and the allowed values Code and Model.

```
<!--Property: implementationType-->
   <oslc:property>
      <oslc:Property>
         <oslc:name>implementationType</oslc:name>
         <oslc:occurs rdf:resource="http://open-service.net/ns/core#
             Exactly-one"/>
         <oslc:propertyDefinition rdf:resource="http://open-services.net
             /ns/iso26262#implementationType"/>
         <oslc:valueType rdf:resource="http://open-services.net/ns/core#
             LocalResource "/>
         <oslc:representation rdf:resource="http://open-services.net/ns/
             core#Inline"/>
         <oslc:range rdf:resource="http://open-services.net/ns/core#Any"
             />
         <oslc:allowedValue rdf:resource="http://open-services.net/ns/
             iso26262#Code"/>
         <oslc:allowedValue rdf:resource="http://open-services.net/ns/
             iso26262#Model"/>
         <oslc:readOnly>true</oslc:readOnly>
      </oslc:Property>
   </oslc:property>
```

Listing 22: Allowed values in ReSh.

In ShEx, allowed values are place in parenthesis after the definition of the property. Listing 23 presents the definition of the property implementationType.

```
#Attribute: implementation type
   shex_iso26262am:implementationType (shex_iso26262am:code
       shex_iso26262am:model),
```

Listing 23: Allowed values in ReSh.

In SHACL, the allowed values can be defined in two ways. One way is defining a class without properties but with instances. This class is see as an enumeration. The instances of this class are used the values of the enumeration, and the allowed values for the property that use the enumeration. For example, the class ImplementationType (which is defined in Listing 24) corresponds with the enumeration that have the same name in the domain SoftwareUnitRealtedConcepts.

```
shacl_iso26262am:ImplementationType
  rdf:type rdfs:Class ;
  rdf:type sh:Shape ;
  rdfs:label "Implementation type"^^xsd:string ;
.
```

Listing 24: Enumeration ImplementationType, defined as a class in SHACL.

Then, instances of the class Implementation Type (defined in Listing 25), which are the values Model and Code, become in the allowed values for the property implementationType (defined in Listing 27)

```
shacl_iso26262am:Code
  rdf:type shacl_iso26262am:ImplementationType ;
  rdfs:label "Code"^^xsd:string ;
.

shacl_iso26262am:Model
  rdf:type shacl_iso26262am:ImplementationType ;
  rdfs:label "Model"^^xsd:string ;
.
```

Listing 25: Instances of the class ImplementationType SHACL.

```
  sh:property [
      sh:class shacl_iso26262am:ImplementationType ;
      sh:description "Describes the type of implementation type used to
          create the software. Allowed values are: Model or Code."^^
          xsd:string;
      sh:maxCount 1 ;
      sh:minCount 1 ;
      sh:name "implementation type"^^xsd:string ;
      sh:nodeKind sh:IRI ;
      sh:predicate shacl_iso26262am:implementationType ;
    ] ;
```

Listing 26: property implementation type in SHACL.

This way to define enumerations gives flexibility to the implementation, since the values of the enumeration are not codified. However, the allowed values can be burned in the code using the constructor *sh:in*. Listing **??** shows the SHACL fragment where the elements are defined using sh:in.

```
  sh:property [
      sh:description "Describes the type of implementation type used to
          create the software. Allowed values are: Model or Code."^^
          xsd:string;
      sh:in (
          shacl_iso26262am:Code
          shacl_iso26262am:model
        ) ;
      sh:maxCount 1 ;
      sh:minCount 1 ;
      sh:name "implementation type"^^xsd:string ;
      sh:predicate shacl_iso26262am:implementationType ;
    ] ;
```

Listing 27: Implementing allowed values in SHACL using *sh:in*.

9. **Requirement 9 - Conditional properties**. SHACL provides the construct *sh:filterShape*, that can be used to limit the scopes of the nodes that are required to be validated. This

characteristic allows the creation filters that meet the requirements defined, like conditionals created with *case*. The Turtle snippet below (Listing 28), shows the shape for a software unit design specification in SHACL, where the ASIL is A, recommended design notation is selected, and thus the design notation rationale becomes mandatory.

```
shacl_iso26262am:DesignNotationShape_AsilARecommendedLevel
  rdf:type sh:Shape ;
  rdfs:label "DesignNotationShape_AsilARecommendedLevel"^^xsd:string ;
  sh:description "ASIL A, recommended notations and  rationale must be
      provided" ;
  sh:filterShape [
      rdf:type sh:Shape ;
      sh:property [
          sh:hasValue shacl_iso26262am:A ;
          sh:predicate shacl_iso26262am:asil ;
        ] ;
      sh:property [
          sh:hasValue shacl_iso26262am:Recommended ;
          sh:predicate shacl_iso26262am:recommendationLevel ;
        ] ;
    ] ;
  sh:property [
      sh:datatype xsd:string ;
      sh:minCount 1 ;
      sh:predicate shacl_iso26262am:designNotationRationale ;
    ] ;
  sh:property [
      sh:in (
          shacl_iso26262am:SemiformalNotations
          shacl_iso26262am:FormalNotations
          shacl_iso26262am:TailoredNotations
        ) ;
      sh:minCount 1 ;
      sh:predicate shacl_iso26262am:designNotationType ;
    ] ;
  sh:scopeClass shacl_iso26262am:SoftwareUnitDesignSpecification ;
.
```

Listing 28: Defining groups of properties in SHACL using *sh:filterShape*.

Define shapes or conditional properties is not possible in ShEx or ReSh

Requirement 10 - Handle RDF collections. This requirement is not possible to implement in ReSh or ShEx. However, SHACL allows for using SPARQL, and the property can be implemented. According to the documentation, one way to define this kind of properties is using the construct *sh:derivedValues*. Unfortunately, the environment used for implementing SHACL specifications does not have execution language ready for this kind of properties. However, it is presented the definition of the property, following documentation principles in Listing 29.

```
shacl_iso26262am:AssignAsil
  rdf:type sh:Shape ;
  rdfs:label "Assign asil"^^xsd:string ;
  sh:property [
      sh:derivedValues [
          rdf:type sh:SPARQLValuesDeriver ;
          sh:select """
    SELECT ?asilImplementation
    WHERE {
     ?this rdf:type shacl_iso26262am:SoftwareUnitImplementation .
        ?softwareUnit shacl_iso26262am:isImplementedBy ?this .
```

```
        ?softwareUnit shacl_iso26262am:asil ?asil .
      BIND (?asil AS ?asilImplementation) .
    }
    """ ;
        ] ;
      sh:maxCount 1 ;
      sh:minCount 1 ;
      sh:predicate shacl_iso26262am:assignedAsil ;
    ] ;
  sh:scopeClass shacl_iso26262am:SoftwareUnitImplementation ;
.
```

Listing 29: Deriving values of properties in SHACL.

6.5.3 Summary of the comparative study related to constraint languages

The constraints defined in Section 6.5.1 where mapped to different three RDF constraint languages (ReSh, ShEx and SHACL) in Section 6.5.2. The result of this mapping is summarized in Table 17.

Table 17: Requirements for RDF constraints mapped to three different RDF data shapes languages.

Requirement	**ReSh**	**ShEx**	**SHACL**
Disjoint properties	No	No	Yes(sh:disjoint)
Minimum qualified cardinality restrictions	No	No	Yes (sh:minCount)
Maximum qualified cardinality restrictions	No	No	Yes (sh:maxCount)
Exact qualified cardinality restrictions	Just exactly-one	No	Yes (using sh:min and sh:max
Cardinality shortcuts	Yes(exactly-one, zero-or-many, zero-or-one, one-or-may)	Yes(using ,?,+,{y,z})	No(but can be reached using sh:min and sh:max)
Inverse object property	No	Yes (using ^)	Yes(sh:inverseProperty)
Context-specific Exclusive Or of properties groups	No	Ye(using operator or)	Yes(sh:or)
Allowed values	Yes (oslc:allowedValues)	Yes(with the values defined in ( )	Yes(sh:in)
Conditional properties	No	No(but can be reacher using Exclusive-OR)	Yes(Using SPARQL or using filter shapes sh:filterShape)
Handle RDF collections	No	No	Yes (using SPARQL queries)

6.6 Discussion

The definition of ISO 26262-compliant OSLC resources shaped with SHACL has required an intense and careful analysis process. Initially, this process involved the identification of the domain, domains terms and domain relationships that address the knowledge needed for the development of this thesis. As it is explained in Section 5.1.1, the elements found in the standard ISO 26262 (specifically the part 6: Software unit design and specification), the OSLC standard (especially the core specification), and Scania practices were considered for the formulation of this domain. Scania practices, in particular, have enriched and given relevance to the domain since industrial best practices are also embraced by the standard ISO 26262, as well as the tailored decisions. As presented in Section 5.1.2, the OSLC domain specifications provided by the OSLC community does not support the needs for the creation of ISO 26262-related resources. The available OSLC domain specifications were created for purposes different to the ones that ISO 26262 is focused on. For this reason, an entirely new OSCL domain that addresses specific vocabulary terms required by ISO

26262:2011 part 6 was created from scratch. This new OSLC domain, which has AM (Architecture Management)-OSLC appearance, but tackles specifically the design and implementation of software units, is called *OSLC_ISO26262AM-SoftwareUnitRelatedConcepts*. The domain specification created also required the use of existing RDF vocabularies. To find the most suitable vocabularies, an analysis on the RDF vocabularies provided by LOV was performed. This analysis, even though was not exhaustive, provides the basic list of vocabularies that were used in the construction of the domain. A more in-depth analysis of RDF languages must be performed in the future, so vocabulary items are not defined again, and the information of the resources is better connected with other information on the Web. Having defined the type of elements required for building the domain, the characteristics of the OSLC domain type required, and the vocabularies to be reused, the next step was the establishment of an approach for converting ISO 26262 elements into OSLC domain specification. The approach used for building the domain specification took elements from the method used in [81]. The purpose of the approach is the translation of the normative parts of the standard into OSLC resources and the addition of the Scania practices. The result obtained after applying the approach was the depiction of the UML-like profile developed in Section 6.1. Once the core domain was sketched, additional constraints (those different to the structural constraints) identified primarily in the standard ISO 26262, were also included. The represented constraints were then mapped to three different but prominent RDF constraint languages namely ReSh, ShEx, and SHACL (see Section 6.5). Other RDF constraint languages could be used for constraining the OSLC resources identified for ISO 26262:2011 part 6. However, the selected ones have precise features that made them good candidates for this study. First, ReSh is the constraint language created specifically for shaping OSLC resources; second, ShEx is a language that has demonstrate, in several comparative studies, its high degree of expressiveness; and third SHACL is a language that compiles the best features found in other constraint languages. Also, SHACL is a new W3C specification from 2016, and it has not been part of other comparative studies, so far. The mapping of the constraints found with the RDF constraint languages selected shows that shapes composed of separately-defined properties in ReSh, make this language limited in its expressiveness. Instead, ShEx and specially SHACL present structures and constructs that allow for defining properties in a more flexible way. ShEx is a language that has its syntax, where the definition of elements requires the use of fewer constructs. This syntax makes the language less verbose, but at the same time, less intuitive. In ReSh, for example, groups of property-values that depend on other property-values can not be specified since ReSh is a language where the shapes are defined as a list of properties. In ShEx, this same property can be expressed using particular operators (exclusive-or, for example), so many properties are linked together inside one shape. SHACL handle this property defining separate constraints, using a construct called *filterShape*. Also, SHACL has specialized constructs that permit the definition of constraint using SPARQL. All these characteristics, as well as remarkable technical features are presented in the tables 8, 9 and 17. In these tables, the information related to the three languages is presented as comparison charts. The observations of these tables allow for understanding the outstanding advantages of using SHACL. So, given this, SHACL is presented as the language that should be preferred for shaping OSLC resources.

A case study is conducted in Chapter 7, to explore the applicability of the domain designed in OSLC and shaped with SHACL. The information presented in this case study is taken from Scania documents and is recalled in Section 2.2.3. The case study aims at providing a model derived form the metadata developed in SHACL.

7 Case study

The solution found in chapter 6 is used in this chapter for conducting the case study. The example used is the software unit *Fuel Level Estimation Algorithm*, recalled in Section 2.2.3. This section is structured in the following way: Section 7.1 presents the elements required for modelling the information of a software unit, Section 7.2 presents the instance of the software unit model, and Section 7.3 provides the validation of the model for the case study.

7.1 Modeling the case study

In this section, the elements required for the case study are identified. Information comes from three different sources. Specific information is recalled from data that is available at Scania. Other information is selected from previous works done at Scania like [13, 3, 18, 97]. Information not recalled from documents is assumed by the author of this thesis. These assumptions are based on documents observations. This subsection is organized as follows: Subsection 7.1.1 presents the data related to the structure of the software unit, Subsection 7.1.2 presents the gathering of ASIL information, and Subsection 7.1.3 presents the gathering of other ISO 26262-related information.

7.1.1 The structure of the software unit

Scania practices are not in compliance with the standard ISO 26262 yet. It means that the information about a software unit, used in this section, is not necessarily safety-related, or at least, not in the sense that is mandated by ISO 26262. However, this information will be used in the creation of the case study, taking into account this limitation. The software unit used is called fuel level estimation algorithm and it is recalled in Section 2.2.3.

7.1.2 ASIL definition

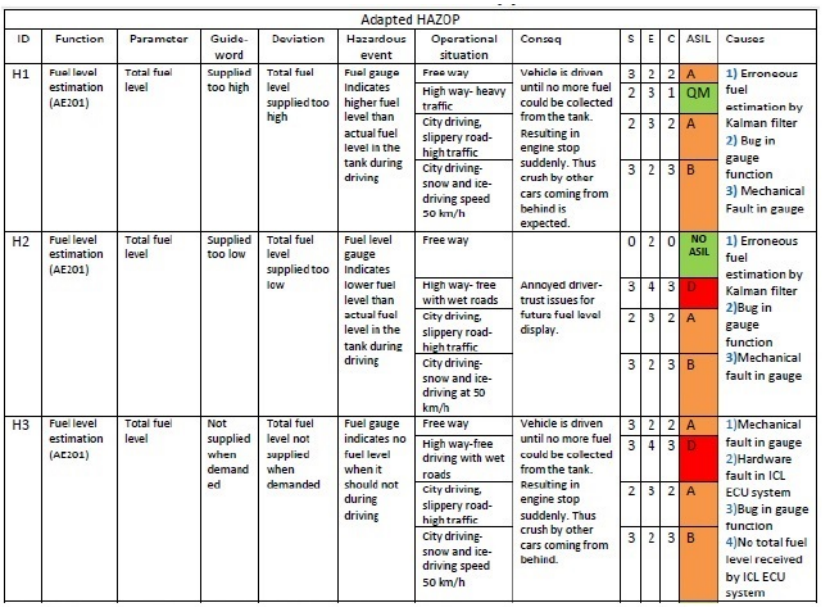
Various research efforts for aligning the safety lifecycle proposed by ISO 26262 to the current practices carried out at Scania, have been done during the last few years. The creation of a safety in compliance with ISO 26262 can be found in [13] and [3]. In order to align the Scania practices, when creating a software unit, with the requirements of the standard, we take a look in these two thesis and recall from them the safety-related information associated with the software unit *fuel level estimation algorithm*. In [13], is stated that *"no hazard analysis and no ASIL classification have been made in Scania, since the standard is not adopted*. However, this evidence was provided by the author of [13], where HAZOP (Hazard and Operability study) was used for hazard analysis and a mapping of the User Functional Requirements (UFRs) found at Scania to the Safety Goals (SGs) required by ISO 26262 was made. For the purpose of this thesis the hazard analysis and Safety goals presented are the ones related with the unit studied in the previous section. This unit is defined in the document AE201 and the HAZOP is presented in Figure 41.

The safety goals defined for the software unit are presented in Figure 42. The ASIL value for both is D. Since ASIL values are propagated during the safety lifecycle and no ASIL decomposition was carried out in [13], the value taken for the purpose of the modeling of this thesis is D.

7.1.3 Other ISO 26262-related information

Other ISO 26262-related elements are required for building the case study. This information is gathered through the observation of the documents already studied. Following it is explained how this information is found.

- **Implementation type**: In Scania, the software design and implementation is carried out in Simulink, using models.
- **Programming Language**: The models created using Simulink are then converted to C code automatically.
- **Design notation type**: According to [98], notations for architecture documentation are the following:



Adapted HAZOP												
ID	Function	Parameter	Guide-word	Deviation	Hazardous event	Operational situation	Conseq	S	E	C	ASIL	Causes
H1	Fuel level estimation (AE201)	Total fuel level	Supplied too high	Total fuel level supplied too high	Fuel gauge indicates higher fuel level than actual fuel level in the tank during driving	Free way	Vehicle is driven until no more fuel could be collected from the tank. Resulting in engine stop suddenly. Thus crush by other cars coming from behind is expected.	3	2	2	A	1) Erroneous fuel estimation by Kalman filter 2) Bug in gauge function 3) Mechanical Fault in gauge
						High way- heavy traffic		2	3	1	QM	
						City driving, slippery road-high traffic		2	3	2	A	
						City driving-snow and ice-driving speed 50 km/h		3	2	3	B	
H2	Fuel level estimation (AE201)	Total fuel level	Supplied too low	Total fuel level supplied too low	Fuel level gauge indicates lower fuel level than actual fuel level in the tank during driving	Free way	Annoyed driver-trust issues for future fuel level display.	0	2	0	NO ASIL	1) Erroneous fuel estimation by Kalman filter 2)Bug in gauge function 3)Mechanical fault in gauge
						High way- free with wet roads		3	4	3	D	
						City driving, slippery road-high traffic		2	3	2	A	
						City driving-snow and ice-driving at 50 km/h		3	2	3	B	
H3	Fuel level estimation (AE201)	Total fuel level	Not supplied when demanded	Total fuel level not supplied when demanded	Fuel gauge indicates no fuel level when it should not during driving	Free way	Vehicle is driven until no more fuel could be collected from the tank. Resulting in engine stop suddenly. Thus crush by other cars coming from behind.	3	2	2	A	1)Mechanical fault in gauge 2)Hardware fault in ICL ECU system 3)Bug in gauge function 4)No total fuel level received by ICL ECU system
						High way-free driving with wet roads		3	4	3	D	
						City driving, slippery road-high traffic		2	3	2	A	
						City driving-snow and ice-driving speed 50 km/h		3	2	3	B	

Figure 41: Hazard Analysis using adapted HAZOP [13].

Safety Goals				
Hazard ID	ASIL	Safety Goal Description	Safety Goal ID	Corresponding UFR
H1, H2	D	The indicated fuel level shall not deviate more than or less than 5% from the actual volume in the tank	SG1	UFR18_1
H3	D	Fuel gauge shall indicate the total fuel level in the tank	SG2	UFR18_8

Figure 42: Safety goals for FLEDS [13].

  - *Informal notations*: Views are depicted using general purpose diagramming and the semantics is characterized in natural language.
  - *Semiformal notations*: Views are expressed in a standardized notation that prescribes graphical elements and rules of construction, like UML.
  - *Formal notations*: Views are described in a notation that has a precise semantics. Formal analysis and generation of code is possible.

  Taking [98] into account, it is found that the software unit design at Scania uses informal notation (the information presented in the documents AE201 and AER 201), and Formal notations (the block diagrams created in Simulink, used then for obtaining code in C).

- **Design principles used**: The design principles required for being in compliance with the standard are listed in Table 3. Pure observations of the models lets us to see that the principle used are the following:

  - *No multiple use of variable names*: The names of the variables are maintained during the creation of the software unit and inside the software functions.
  - *No recursion.* There are not recursive functions in the model.

7.2 Case study model

This section is organized in the following way: Subsection 7.2.1 presents the case study information, organized in an UML Object diagram, and Subsection 7.2.2 presents the information of the case

study in an RDF model constrained by SHACL.

7.2.1 UML Object diagram

The elements identified in Section 7.1 are modeled in a UML object diagram. This modeling elements are presented in the following five items.

1. *Software unit design specification object*: To instantiate the SoftwareUnitDesignSpecfication class the information required is the *ASIL*, which was defined in Subsection 7.1.2 as *"D"*. Other information, defined in Subsection 7.1.3, are: *implementationType: "Model"*, *designNotationtype: "Informal notations"* and *"Formal notations"*, and *DesignPrincipleSelected: "No multiple use of variable names"* and *"No recursion"*. Since the design notation types and the design notation principles corresponds with the ASIL selected for the software unit design, the attributes *designNotationRationale* and *designPrincipleSelectedRationale* are not mandatory. Information about *functionalBehavior* and *description* were found in Section 7.1.2 and they are defined as *"Software unit in charge of calculating the current percentage level in the fuel tank"* for the first attribute, and *"Software unit composed of several functions and implemented as a defined user block in Simulink"* for the second attribute. A summarize of the instance information is presented in Table 18. The instance of the class is presented in Figure 43.

Table 18: Data required for instantiate the class SoftwareUnitDesignSpecification.

Attribute	**Data**
asil	D
implementationType	Model
designNotationType	Informal Notations
designNotationType	Formal Notations
functionalBehaviour	Software unit in charge of calculating the current percentage level in the fuel tank.
description	Software unit composed of several functions and implemented as a defined user block in Simulink.
designPrincipleSelected	No multiple use of variable names.
designPrincipleSelected	No recursion

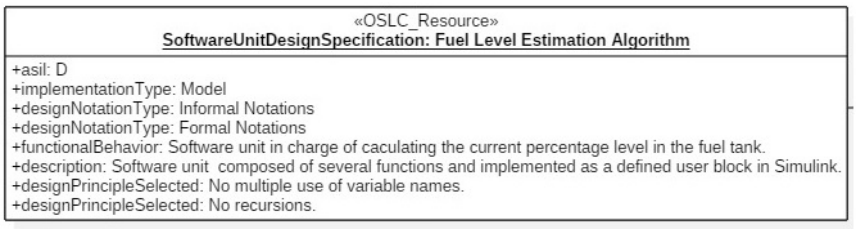


Figure 43: SoftwareUnitDesignSpecification instance: Fuel Level Estimation Algorithm.

2. *Software unit implementation object*: To instantiate the SoftwareUnitImplementation class, the information required is the *ASIL*, which is inherited from the ASIL of the corresponding software unit design specification, and its value is "D"; *programmingLanguage*, which was defined in Section 7.1.3 as "C"; and the *designPrinciple selected*, the same used in the corresponding software unit design specification, since the model, at Scania, represent both the design and the implementation. A summarize of this information is presented in Table 19 and the object instantiation is presented in Figure 44.

3. *Software unit function object*: The software unit function instantiated in this model is *CalculatCurrentVolumeLevels*. The information required for doing this instance is the *description*,

Table 19: Data required for instantiate the class SoftwareUnitImplementation.

Attribute	**Data**
asil	D
programmingLanguage	C
designPrincipleSelected	No multiple use of variable names.
designPrincipleSelected	No recursion

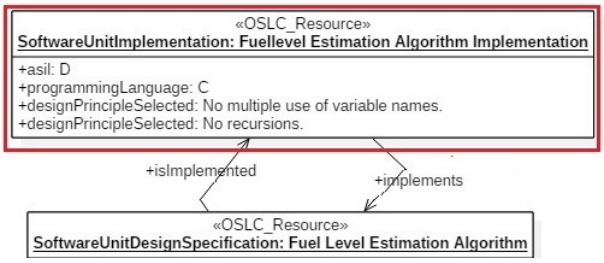


Figure 44: SoftwareUnitImplementation instance: Fuel Level Estimation Algorithm.

which is found in Section 7.1.1. This software Unit function has two subfunctions. The information required for the instantiate is summarized in Table 20, and the instance is depicted in Figure 45.

Table 20: Data required for instantiate the class SoftwareUnitFunction.

Attribute	**Data**
function: CalculatCurrentVolumeLevels	
description	This software unit function is in charge of mapping the fuel level sensor depending on the tank variant, and defining the x0 state which is the startup state of the Kalman filter.
subfunction: Precalculations	
description	In charge of doing some precalculations required for finding the measured levels
subfunction: CalculateX0	
description	In charge of the startup state for the Kalman filter algorithm.

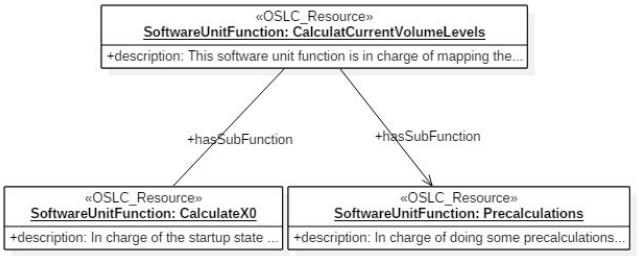


Figure 45: SoftwareUnitFunction instance: CalculatCurrentVolumeLevels.

4. *Variables* The variables related with the software unit design specification are described in Section 7.1.1. A summarize of the information required for the instance is presented in Table 21, and the instantiation of this variables is presented in Figure 46. The complete object diagram elements, including configuration parameters and parameter values is depicted in figure 47.

Table 21: Data for the input and output ports.

Attribute	**Data**
input port: fuel_params_inputBus_str	
description	required to storage the fuel parameters values stored in the RTD
dataType	String
unit	Not specified
input port: fuel_inputBus_str	
description	Fuel values obtained from the fuel level sensors and stored in the RTD.
dataType	String
unit	Not specified
input port: tankCapacity_str	
description	Read the capacity of the fuel tank.
dataType	String
unit	L
Required range	0 to 2500
Required accuracy	1
output port: fuelVolume_str	
description	Estimated total fuel level converted to liters
dataType	String
unit	L
output port: fuelLevelTot_str	
description	Estimated total fuel level in the tank
dataType	String
unit	%
Required range	0 to 100
Required accuracy	0.4
output port: reset_str	
description	It is a variable that provide information about the good status of the signals.
dataType	String
unit	Not specified

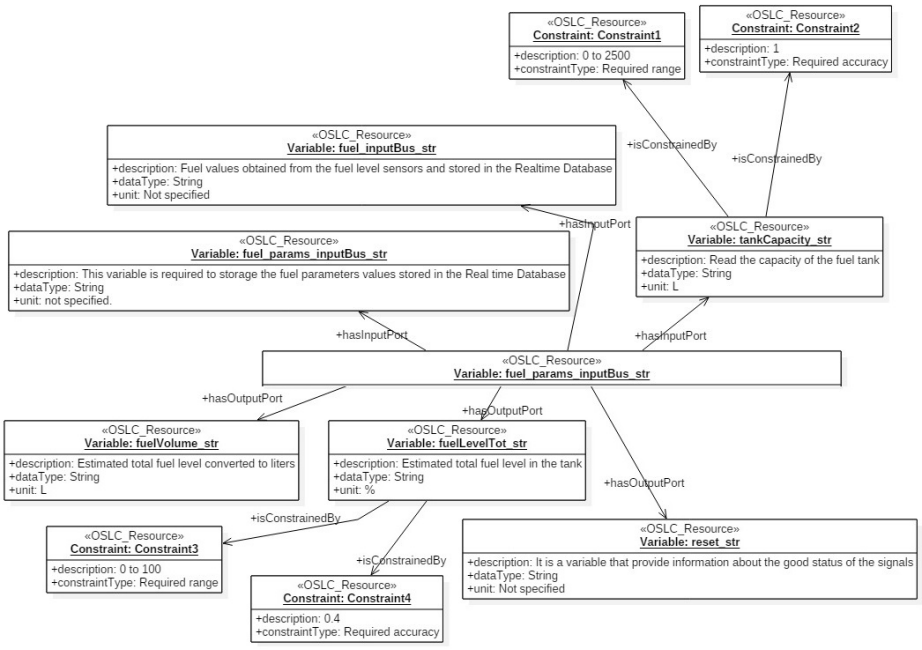


Figure 46: Variable instances: Input port objects.

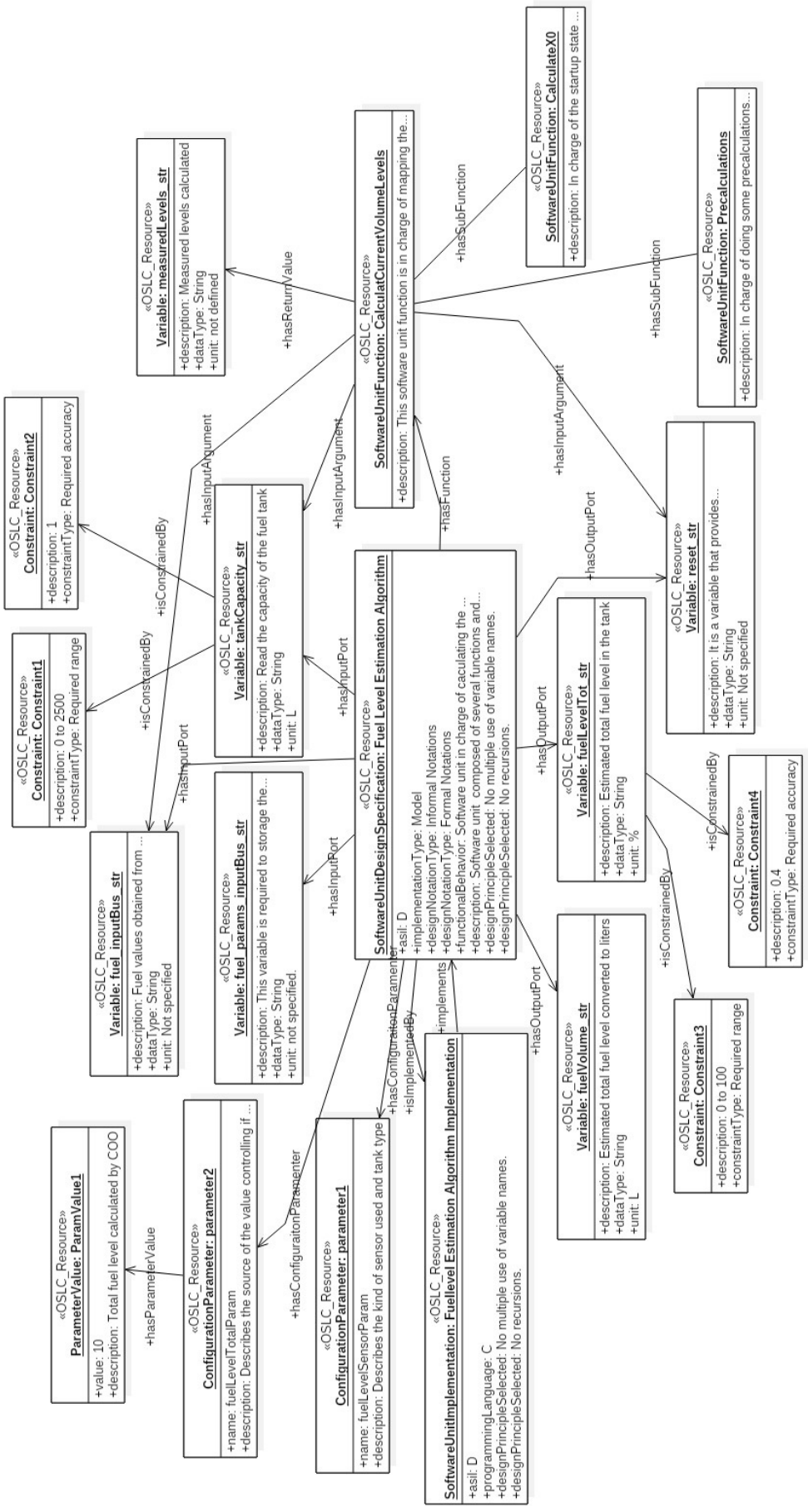


Figure 47: Representation of a the software unit design and implementation information.

7.2.2 Case study data constrained with SHACL

Using the editor *TopBraid Composer*, the instances objects presented in Section 7.2.1 are easily adhered to a RDF model that is being constrained by SHACL.

1. **Creating the enumerations**: For the modeling, six enumerations (depicted in Figure 39) are required, namely *ASIL*, *SoftwareUnitDesignPrinciple*, *RecommendationLevel*, *SoftwareUnitDesignNotation*, *ConstraintType* and *ImplementationType*. For exemplification purposes, the methodology to add the enumerations used in this project is explained with the *ASIL* enumeration. In the composer, a shape with the name ASIL is created. No constraints, scopes or properties are added to this shape. The created shape is populated with the instances, namely *A*, *B*, *C*, *D* and *QM*. Figure 48 shows the composer with the shape ASIL created and populated with the instances.

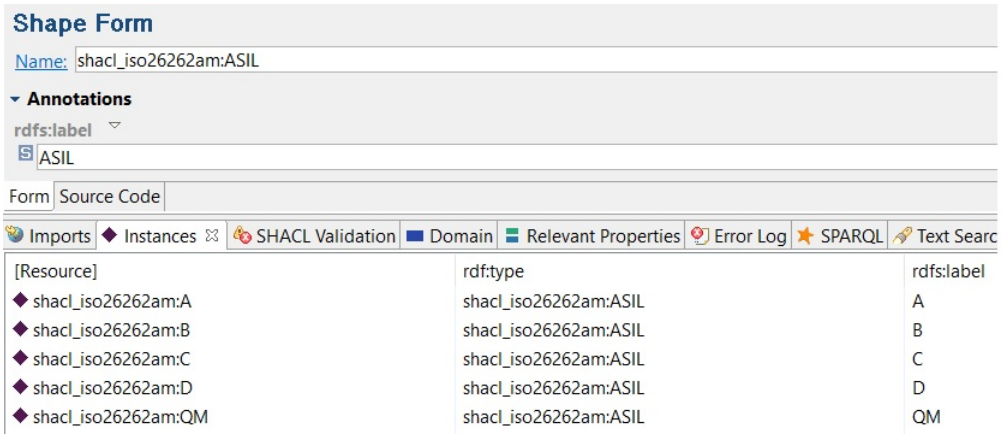


Figure 48: Enumeration ASIL in SHACL (created in TopBraid Composer).

2. **Adding data to the classes**: In Figure 49 is presented the specification for the class SoftwareUnitDesignSpecification, in SHACL, using the tool TopBraid Composer.

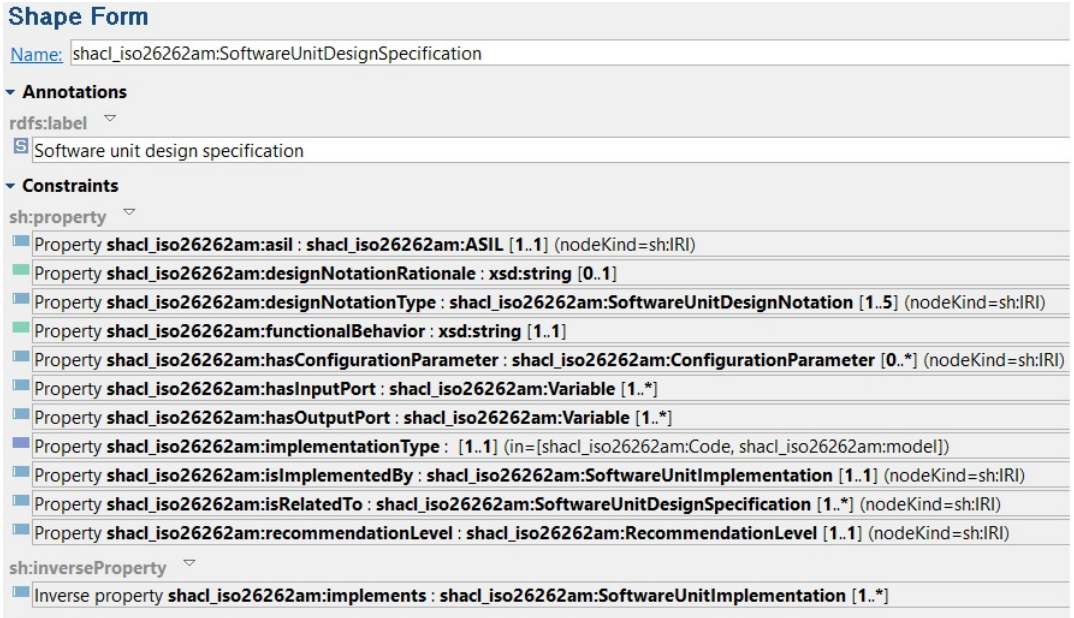


Figure 49: Software Unit Design Specification shape in SHACL (created in TopBraid Composer).

Seven classes have been modeled in Figure 39, namely *SoftwareUnitDesignSpecification*, *SoftwareUnitImplementation*, *SoftwareUnitFunction*, *Variable*, *Constraint*, *ConfigurationParam-*

eter and *ParameterValue*. All this classes has also been created using the tool *TopBraid composer*. Once the classes are created, the instances can also be added. TopBraid connects all the information and shows if the constraints are violated. The form that contains the data for Fuel Level Estimation Algorithm is presented in Figure 50.

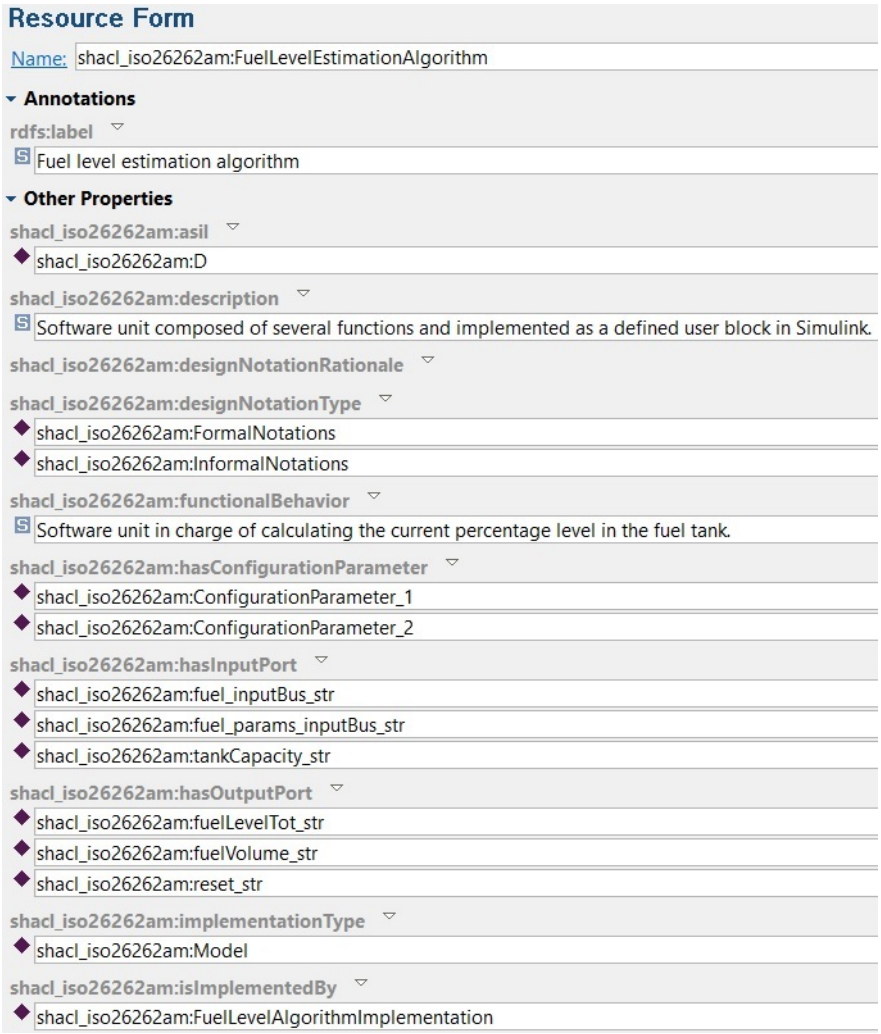


Figure 50: Data for Fuel Level Estimation Algorithm in SHACL (created in TopBraid Composer).

The RDF (serialized with XML/RDF) generated by TopBraid is presented in Listing 30.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:shacl_iso26262am="http://http://open-services.net/ns/
      shacl_iso26262am#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <rdf:Description rdf:about="http://http://open-services.net/ns/
      shacl_iso26262am#FuelLevelEstimationAlgorithm">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Fuel level estimation algorithm</rdfs:label>
    <shacl_iso26262am:hasConfigurationParameter rdf:resource="http://
        http://open-services.net/ns/shacl_iso26262am#
        ConfigurationParameter_2"/>
```

```
    <shacl_iso26262am:isImplementedBy rdf:resource="http://http://open-
        services.net/ns/shacl_iso26262am#
        FuelLevelAlgorithmImplementation"/>
    <shacl_iso26262am:asil rdf:resource="http://http://open-services.net
        /ns/shacl_iso26262am#D"/>
    <shacl_iso26262am:hasInputPort rdf:resource="http://http://open-
        services.net/ns/shacl_iso26262am#fuel_params_inputBus_str"/>
    <shacl_iso26262am:hasInputPort rdf:resource="http://http://open-
        services.net/ns/shacl_iso26262am#tankCapacity_str"/>
    <shacl_iso26262am:hasInputPort rdf:resource="http://http://open-
        services.net/ns/shacl_iso26262am#fuel_inputBus_str"/>
    <shacl_iso26262am:hasOutputPort rdf:resource="http://http://open-
        services.net/ns/shacl_iso26262am#reset_str"/>
    <dcterms:description rdf:datatype="http://www.w3.org/2001/XMLSchema#
        string">Software unit  composed of several functions and
        implemented as a defined user block in Simulink.</
        dcterms:description>
    <shacl_iso26262am:designNotationType rdf:resource="http://http://
        open-services.net/ns/shacl_iso26262am#InformalNotations"/>
    <shacl_iso26262am:implementationType rdf:resource="http://http://
        open-services.net/ns/shacl_iso26262am#Model"/>
    <shacl_iso26262am:hasFunction rdf:resource="http://http://open-
        services.net/ns/shacl_iso26262am#CalculatCurrentVolumeLevels"/>
    <shacl_iso26262am:hasOutputPort rdf:resource="http://http://open-
        services.net/ns/shacl_iso26262am#fuelLevelTot_str"/>
    <shacl_iso26262am:hasConfigurationParameter rdf:resource="http://
        http://open-services.net/ns/shacl_iso26262am#
        ConfigurationParameter_1"/>
    <shacl_iso26262am:designPrincipleSelected rdf:resource="http://http:
        //open-services.net/ns/shacl_iso26262am#
        No_multiple_use_of_variable_names"/>
    <rdf:type rdf:resource="http://http://open-services.net/ns/
        shacl_iso26262am#SoftwareUnitDesignSpecification"/>
    <shacl_iso26262am:recommendationLevel rdf:resource="http://http://
        open-services.net/ns/shacl_iso26262am#Recommended"/>
    <shacl_iso26262am:designNotationType rdf:resource="http://http://
        open-services.net/ns/shacl_iso26262am#FormalNotations"/>
    <shacl_iso26262am:designPrincipleSelected rdf:resource="http://http:
        //open-services.net/ns/shacl_iso26262am#No_recursions"/>
    <shacl_iso26262am:hasOutputPort rdf:resource="http://http://open-
        services.net/ns/shacl_iso26262am#fuelVolume_str"/>
    <shacl_iso26262am:functionalBehavior rdf:datatype="http://www.w3.org
        /2001/XMLSchema#string">Software unit in charge of calculating
        the current percentage level in the fuel tank.</
        shacl_iso26262am:functionalBehavior>
  </rdf:Description>

</rdf:RDF>
```

Listing 30: RDF data for the Software Unit Fuel Level Estimation Algorithm.

Figure 51 presents the RDF graphs that corresponds with the XML/RDF fragment presented in Listing 30. This graph is created with the W3C RDF validation service[19].

[19] https://www.w3.org/RDF/Validator/

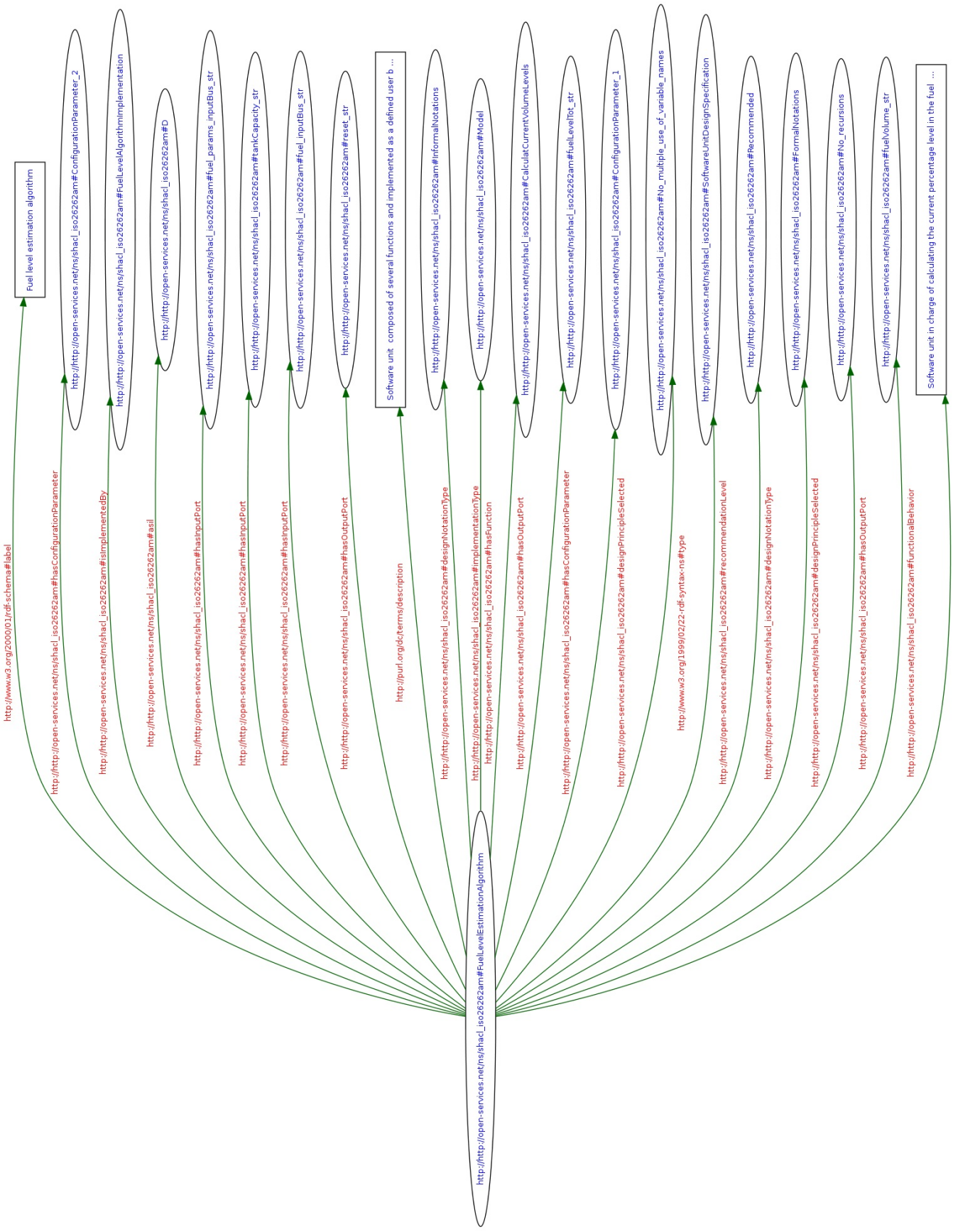


Figure 51: RDF graphs representation of XML/RDF fragment presented in Listing 30.

7.3 Model Validation

The model generated in Figure 51 presents the safety-related information for the software unit Fuel Level Estimation Algorithm. This model was automatically generated from a SHACL-defined metamodel. This section is presenting validation mechanisms, so the information generated corresponds with the ISO 26262 requirements for software unit design specification. This section is structured in the following way: Subsection 7.3.1 presents the validation of the ISO 26262 requirements in the model, and Subsection 7.3.2 presents the method used for the constraints validation.

7.3.1 ISO 26262 requirements validation

To verify the completeness of the model generated, a set of questions, based on the standard ISO 26262, have to be answered. These questions are formulated, taking into account the requirements for software unit design specification presented in the standard (these requirements, as recalled in Section 2.3.3 corresponds with the numerals 8.4.2 to 8.4.4). The questions are presented in Table 22.

Table 22: Questions related with the ISO 26262 requirements for software unit design specification.

Requirement	Question
8.4.2	Is there information that express if the software unit specified is safety-related? Is there information that communicate the design notations used for describing the software unit?
8.4.3	Is the functional behavior of the software unit described? Is the internal design of the software units described?
8.4.4	Are there design principles associated with the software unit design specification?

The questions are answered as follows:

1. **Is there information that express if the software unit specified is safety-related?**

   Yes, there is information that specify if the software unit design specification is safety-related. This information is defined in the model as *shacl_iso26262am:asil*, which value in the instance is *http://open-services.net/ns/shacl_iso26262am#D*

2. **Is there information that communicate the design notations used for describing the software unit?**

   Yes, There is information that communicate the design notations used for describing the software units. This information is defined in the model as *shacl_iso26262am:designNotationType*, which values in the instance are: *http://open-services.net/ns/shacl_iso26262am#InformalNotations* and *http://open-services.net/ns/shacl_iso26262am#FormalNotations*

3. **Is the functional behavior of the software unit described?**

   Yes, the functional behavior of the software unit is described. This information is defined in the model as *shacl_iso26262am:functionalBehavior*, which value in the instance is the string *"Software unit in charge of calculating the current percentage level in the fuel tank."*

4. **Is the internal design of the software units described?**

   Yes, there are characteristics of software unit that describe its internal design. These characteristics and their values are:

   - *rdfs:label*:Fuel level estimation algorithm.
   - *dcterms:description*:Software unit composed of several functions and implemented as a defined user block in Simulink.
   - *shacl_iso26262am:hasInputPort*: `http://open-services.net/ns/shacl_iso26262am#fuel_params_inputBus_str`

- *shacl_iso26262am:hasInputPort*: `http://open-services.net/ns/shacl_iso26262am#fuel_inputBus_str`
- *shacl_iso26262am:hasInputPort*: `http://open-services.net/ns/shacl_iso26262am#tankCapacity_str`
- *shacl_iso26262am:hasOutputPort*: `http://open-services.net/ns/shacl_iso26262am#fuelVolume_str`
- *shacl_iso26262am:hasOutputPort*: `http://open-services.net/ns/shacl_iso26262am#reset_str`
- *shacl_iso26262am:hasOutputPort*: `http://open-services.net/ns/shacl_iso26262am#fuelLevelTot_str`
- *shacl_iso26262am:hasFunction*: `http://open-services.net/ns/shacl_iso26262am#CalculatCurrentVolumeLevels`
- *shacl_iso26262am:hasConfigurationParameter*: `http://open-services.net/ns/shacl_iso26262am#ConfigurationParameter_1`

5. **Are there design principles associated with the software unit design specification?**

   Yes, there are design principles associated with the software unit specified. This information is defined in the model as *http://http://open-services.net/ns/shacl_iso26262am#No_multiple_use_of_variable_names* and `http://http://open-services.net/ns/shacl_iso26262am#No_recursions`

7.3.2 Constraints validation

To verify the constraint formulated in Section 6.5 is required tool support. Since the tool used to generate the SHACL specification was TopBraid Composer, this same tool is in charge of verifying the constraints. It helped in the verification of the majority of the constraint in an automatic way. Just one constraint, defined in Section 6.5 as *Disjoint properties* did not have yet support in this tool. An example of this verification can be seen in Figure 52 where a ! symbol is placed at the end of the property that is missing a value. Other way in which TopBraid shows error messages is in its SHACL validation box with messages like the one in Figure 53.

Figure 52: Constraint Verification (defined in TopBraid Composer).

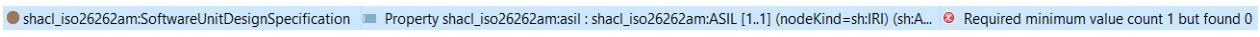


Figure 53: Validation Message (defined in TopBraid Composer).

The software unit modeled did not required the association of *"rationale"* fields, like designNotationRationale or designPrincipleSelectedRationale since all the information directly complaint with the ASIL assigned. However, if *ASIL* and other fields, like *designNotationType* or *implementationType* did not correspond, the field *designNotationRationale* must be activated as mandatory, with the help of the tool TopBraid Composer.

8 Conclusions

This chapter present the final thoughts of the job elaborated in this master thesis. It is organized as follows: In Section 8.1 the summary of the outcomes of the thesis work is presented, and in Section 8.2 directions for future actions are described.

8.1 Summary

This thesis has explored means to mitigate the process of creating an ISO 26262-compliant safety case, where product-based evidence is provided. The data management is supported by the definition of an OSLC domain specification that addresses a tiny portion of the standard. This infrastructure aims at providing the traceability required when different tools are used for the documentation of the safety lifecycle. So, tools that addresses requirements specification, architecture specification, among others, can cooperate in a seamless way. Specifically, this thesis consisted in the identification, representation, and shaping of resources needed to create ISO 26262-compliant safety case, with the provision of software unit design and implementation information. The focus was limited to a small portion of ISO 26262, but the results can be extended to other parts of the standard. The portion corresponds to the left-hand side of the V-model for product development at the software level (ISO 26262-6:2011): *software unit design and specification.*

To enable the creation of the OSLC domain, the first step was the identification of the right resources to be exchanged. During the identification stage, elements required for building a domain are analyzed. For this, the approach discussed by [99] was also adopted in this thesis. This approach included a careful analysis of the standard, as well as the documentation provided by Scania. The second step was the representation of these resources. For the resources representation, elements identified are characterized and depicted in a UML-like model. UML profile was selected for modeling, since Scania practices have adopted this modeling support technique, and the results of this thesis have to be aligned with the practices carried out by the company. For representing the already identified resources was necessary to determine if an OSLC existing specification domain could be use or extended. It was found that the current versions of the OSLC domain specification could not support the requirements of the ISO 26262-related resources. So, in the UML profile was included the definition of a new AM-like OSLC-domain called *OSLC_ISO26262AM-SoftwareUnitRelatedConcepts* that targets ISO 26262-6. Finally, throughout the shaping stage, the constraints found for describing and enhancing the quality of the resources, are specified and developed in three prominent languages. Shaping domains specification ease the process of collecting information since it guides the user in the addition of the correct information. These constraints came from a second reading and more profound analysis of the standard ISO 26262. Once the constraints were established, a comparative study of three prominent RDF constraint languages (ReSh, ShEx and SHACL) was performed in order to find a good candidate for shaping the ISO 26262-related work products. The RDF constraint language that shows more expressivity and better technical support was SHACL. So, in the framework of this thesis, SHACL is considered the best alternative for shaping the RDF resources required. Taking this into account, metadata in SHACL was specified, and an instance from a case study extracted from Scania, the software unit: *fuel level estimation algorithm*, was generated. The metadata specification in SHACL was created using the tool Top Braid Composer, which successfully produce well-formed XML resources, and could validate the majority of the constraints defined. The information represented in the resource originated have shown compliance with the requirements of the ISO 26262 for software unit design specification.

8.2 Future work

For certification purposes, or in the case of ISO 26262-compliant assessment, it is important to provide process-based arguments [91] and product-based arguments [100] that shows that ISO 26262-compliant process activities have been performed (or tailored appropriately). The provision of these arguments can be done in a (semi)automatic way, so the human reasoning can be supported. This (semi)automatic generation of argument-fragments enables the reduction of the efforts done by safety engineers when creating safety arguments. The work presented in this thesis enables part

of the generation of the argument-fragments, but more more steps are needed, so the provision of an ISO 26262-compliant safety case can be completely supported. The steps that are though as future job are the following:

- As proposed by [81], similar knowledge representation for product-based evidence, with the definition of constraints to validate and guarantee the quality of data provided by the models, can be extended to other parts of the standard ISO 26262.
- Process-based evidence can also be analyzed, so metamodels for supporting this king of argumentation can be defined.
- The provision of traceability between both sides of the V-model can be analyzed, merging the results of this work (centered on the left side of the V-model) with the work [81] centered on the right-hand side of the V-model.
- Further, query mechanisms for retrieving information from resources, and the creation of an argument-fragment from the resources retrieved, can be exploited.

References

[1] "VINNOVA." [Online]. Available: http://www.vinnova.se/sv/resultat/projekt/effekta/espresso/

[2] "Gen&ReuseSafetyCases-SSF." [Online]. Available: http://www.es.mdh.se/projects/393-genreusesafetycases

[3] A. Gallucci, "Building a safety case for a small sized product line of Fuel Level Display Systems," Master thesis, Mälardalen University, 2013.

[4] R. Agrawal, "Semi-Automated Formalization and Verification of Automotive Requirements using Simulink Design Verifier," Master Thesis, KTH, 2015.

[5] "ISO 26262. Road vehicles Functional safety." 2011.

[6] J. M. Alvarez-rodríguez, J. Llorens, M. Alejandres, and J. Miguel, "OSLC - KM : A knowledge management specification for OSLC based resources," 2015.

[7] W3C, "RDF 1.1 Concepts and Abstract Syntax," 2014. [Online]. Available: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[8] A. Ryman, "Resource Shape 2.0," 2014. [Online]. Available: https://www.w3.org/Submission/shapes/

[9] H. Knublauch and A. Ryman, "Shapes Constraint Language (SHACL)," 2016. [Online]. Available: https://www.w3.org/TR/2016/WD-shacl-20160128/

[10] D. Johnson and S. Speicher, "Open Services for Lifecycle Collaboration Core Specification Version 2.0," 2013. [Online]. Available: http://open-services.net/bin/view/Main/OslcCoreSpecification

[11] J. W. Creswell, *Research Design: Qualitative, Quantitative and Mixed Methods Approach*, 4th ed. Lincoln, USA: Sage Publications Inc, 2014.

[12] "Linked Open Vocabularies (LOV)," 2016. [Online]. Available: http://lov.okfn.org/dataset/lov

[13] R. Dardar, "Building a Safety Case in Compliance with ISO 26262," Master Thesis, Mälardalen University, 2013.

[14] E. Prudhommeaux, "Shape Expressions (ShEx) Primer," 2016. [Online]. Available: http://shexspec.github.io/primer/

[15] P.-y. Vandenbussche, G. A. Atemezing, B. Vatant, and M. Poveda-Villalón, "Linked Open Vocabularies (LOV): a gateway to reusable semantic vocabularies on the Web," *Semantic web journal '16*, vol. 1, p. 17, 2016.

[16] D. Ward, "ISO 26262 Update on development of the standard," no. January, pp. 1–11, 2016.

[17] B. Gallina, J. P. Castellanos Ardila, and M. Nyberg, "Towards Shaping ISO 26262-compliant Resources for OSLC-based Safety Case Creation," in *Critical Automotive applications: Robustness & Safety (CARS)*, Göteborg, Sweden, 2016, p. 4.

[18] R. Dardar, B. Gallina, A. Johnsen, K. Lundqvist, and M. Nyberg, "Industrial experiences of building a safety case in compliance with ISO 26262," *Proceedings - 23rd IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW 2012*, pp. 349–354, 2012.

[19] ISO, "Systems and software engineering Vocabulary," 2010. [Online]. Available: https://www.iso.org/obp/ui/

[20] B. Gallina and M. Nyberg, "Reconciling the ISO 26262-compliant and the Agile Documentation Management in the Swedish Context." in *Proceedings of the third Workshop on Critical Automotive applications: Robustness & Safety, joint event of EDCC-2015*, Paris, 2015.

[21] "Open Services for Lifecycle Collaboration," 2013. [Online]. Available: http://open-services.net/wiki/core/specification-3.0

[22] Scania, "About Scania," 2015. [Online]. Available: http://scania.se/om-scania/

[23] "ECSEL AMASS project." [Online]. Available: http://www.amass-ecsel.eu/

[24] L. C. Avizienis Algirdas, Laprie Jean-Claude, Randell Brian, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[25] D. Smith and K. Simpson, *Safety Critical Systems Handbook*. Butterworth-Heinemann, 2010.

[26] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival, "A Static Analyzer for Large Safety-Critical Software," *Pldi*, pp. 196–207, 2003.

[27] N. Silva and M. Vieira, "Towards Making Safety-Critical Systems Safer : Learning from Mistakes," 2014.

[28] J. C. Knight, "Safety critical systems: challenges and directions," *Proceedings of the 24rd International Conference on Software Engineering (ICSE), 2002. IEEE.*, pp. 547 – 550, 2002.

[29] S. Nair, J. L. De La Vara, A. Melzi, G. Tagliaferri, L. De-La-Beaujardiere, and F. Belmonte, "Safety evidence traceability: Problem analysis and model," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8396 LNCS, pp. 309–324, 2014.

[30] A. Algirdas, L. Jean-Claude, R. Brian, and L. Carl, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[31] L. Feinbube, P. Tröger, and A. Polze, "The landscape of software failure cause models," *arXiv:1603.04335*, 2016.

[32] E. S. Grant, V. K. Jackson, and S. A. Clachar, "Towards a Formal Approach to Validating and Verifying Functional Design for Complex Safety Critical Systems," *GSTF Journal on Computing*, vol. 2, no. 1, pp. 202–207, 2012.

[33] IEC, "Functional safety - Essential to overall safety," *IEC Functional safety 2015-03*, p. 12, 2015.

[34] N. J. Bahr, *System Safety Engineering and Risk Assessment A Practical Approach*. CRC Press, 2015.

[35] R. Faragher, "Understanding the Basis of the Kalman Filter via a Simple and Ituitive Derivation," *IEEE Signal Processing Magazine*, no. September, pp. 128–132, 2012.

[36] F. Sundberg, "Function Allocation Description FAD-UF18 Fuel Level Display," SCANIA, Tech. Rep., 2015.

[37] ISO, "Road vehicles Functional safety Part 1: Vocabulary," 2011. [Online]. Available: https://www.iso.org/obp/ui/

[38] J. Rakos, K. Dhanraj, S. Kennedy, L. Fleck, S. Jackson, and J. Harris, *The Practical Guide to Project Management Documentation*. John Wiley & Sons, 2015.

[39] C. J. Satish and M. Anand, "Software Documentation Management Issues and Practices: A Survey," *Indian Journal of Science and Technology*, vol. 9, no. 20, 2016.

[40] I. Habli and T. Kelly, "Process and product certification arguments- Getting the balance right," *ACM SIGBED Review*, vol. 3, no. August 2016, pp. 1–8, 2006.

[41] M. Bender, T. Maibaum, M. Lawford, and A. Wassyng, "Positioning Verification in the Context of Software / System Certification," *11th International Workshop on Automated Verification of Critical Systems (AVoCS 2011)*, vol. 46, p. 15, 2011.

[42] H. Doerr and I. Stuermer, "Managing an ISO 26262 Safety Case: A Software System Perspective," SAE, Tech. Rep., 2016.

[43] IEEE, "IEEE-Standards Glossary." [Online]. Available: https://www.ieee.org/education_careers/education/standards/standards_glossary.html

[44] ISO, "Semantic interoperability of health information." [Online]. Available: http://www.en13606.org/the-ceniso-en13606-standard/semantic-interoperability

[45] W. Gödert, M. Nagelschmidt, and J. Hubrich, *Semantic Knowledge Representation for Information Retrieval*, 2014.

[46] M.-A. Sicilia, "Metadata research: Making digital resources useful again?" in *Handbook of metadata, semantics and ontologies*, M.-A. Sicilia, Ed. Madrid: World Scientific Publishing Co., 2014, ch. I.1, pp. 1–8.

[47] S. J. Mellor and M. J. Balcer, *Executable UML. A foundation for Model-Driven Architecture.* Addison Wesley, 2002.

[48] A. B. Markman, *Knowledge representation.* Psychology Press, 2013.

[49] M. Seidl, M. Scholz, C. Huemer, and G. Kappel, *UML @ Classroom.* Springer, 2012.

[50] W3C, "W3C Internationalization," 2011. [Online]. Available: https://www.w3.org/International/O-URL-and-ident.html

[51] "XML RDF." [Online]. Available: http://www.w3schools.com/xml/xml{_}rdf.asp

[52] "Reasoner." [Online]. Available: https://www.w3.org/2001/sw/wiki/Category:Reasoner

[53] Patrick J. Hayes and P. F. Patel-Schneider, "RDF 1.1 Semantics," 2014. [Online]. Available: https://www.w3.org/TR/rdf11-mt/

[54] "Serialización (C# y Visual Basic)." [Online]. Available: https://msdn.microsoft.com/es-es/library/ms233843.aspx

[55] "RDF 1.1 Turtle." [Online]. Available: https://www.w3.org/TR/turtle/

[56] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," 2005. [Online]. Available: http://www.ietf.org/rfc/rfc3986.txt

[57] W3Schools, "XML Tutorial." [Online]. Available: http://www.w3schools.com/xml/

[58] T. Bray, D. Hollander, and A. Layman, "Namespaces in XML," 199. [Online]. Available: https://www.w3.org/TR/1999/REC-xml-names-19990114/

[59] W3Schools, "XML RDF," 2016. [Online]. Available: http://www.w3schools.com/xml/xml_rdf.asp

[60] T. Berners-Leer, "Linked Data," 2009. [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html

[61] S. Decker, F. V. Harmelen, J. Broekstra, M. Erdmann, D. Fensel, I. Horrocks, M. Klein, and S. Melnik, "The Semantic Web - on the respective Roles of XML and RDF," *IEEE Internet Computing*, vol. 4, no. October, p. 19, 2000.

[62] "Semantic web," *W3C*, 2015. [Online]. Available: https://www.w3.org/standards/semanticweb/

[63] J. Domingue, D. Fensel, and J. A. Hendler, "Semantic Web Architecture," in *Handbook of Semantic Web Technologies*. Springer Science & Business Media, 2011, ch. 1, pp. 1–41.

[64] S. Powers, *Practical RDF*. OReilly Media, 2003.

[65] "RDF 1.1 Primer," 2014. [Online]. Available: https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/

[66] "RDF Schema 1.1," 2014. [Online]. Available: https://www.w3.org/TR/rdf-schema/

[67] D. Brickley and R. Guha, "RDF Schema 1.1," 2014. [Online]. Available: https://www.w3.org/TR/rdf-schema/

[68] "SPARQL 1.1 Query Language." [Online]. Available: https://www.w3.org/TR/sparql11-query/

[69] B. DuCharme, *Learning SPARQL*. O'Reilly Media, Inc., 2013.

[70] E. Prud'hommeaux, J. E. Labra Gayo, and H. Solbrig, "Shape expressions: An RDF validation and transformation language," *Proceedings of the 10th International Conference on Semantic Systems - SEM '14*, pp. 32–40, 2014.

[71] W3C, "RDF Data Shapes Working Group," 2016. [Online]. Available: https://www.w3.org/2014/data-shapes/wiki/Main_Page#RDF_Data_Shapes_Working_Group

[72] T. Bosch and K. Eckert, "Guidance , Please! Towards a Framework for RDF-based Constraint Languages ." in *International Conference on Dublin Core and Metadata Applications*, 2015, p. 19.

[73] A. Ryman, A. Le Hors, and S. Speicher, "OSLC Resource Shape: A language for defining constraints on Linked Data," *LDOW*, no. 996, 2013.

[74] "OSLC Core Version 3.0. Part 6: Resource Shape." [Online]. Available: http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csprd01/part6-resource-shape/oslc-core-v3.0-csprd01-part6-resource-shape.html

[75] D. Johnson, "Expressing relationships for OSLC," 2012. [Online]. Available: http://open-services.net/bin/view/Main/OslcCoreSpecAppendixLinks

[76] W3C, "Linked Data," 2015. [Online]. Available: https://www.w3.org/standards/semanticweb/data

[77] "Open Services for Lifecycle Collaboration." [Online]. Available: http://open-services.net/

[78] S. Speicher and J. El-khoury, "An Introduction to OSLC and Linked Data." [Online]. Available: http://open-services.net/linked-data-and-oslc-tutorial-2015-update/

[79] J. L. De La Vara and R. K. Panesar-Walawege, "SafetyMet: A metamodel for safety standards," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8107 LNCS, pp. 69–86, 2013.

[80] Y. Luo, M. Van Den Brand, L. Engelen, J. Favaro, M. Klabbers, and G. Sartori, "Extracting models from ISO 26262 for reusable safety assurance," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7925 LNCS, pp. 192–207, 2013.

[81] K. Padira, "Investigation of Resources Types for OSLC domains Targeting ISO 26262," Master Thesis, Blekinge University, 2016.

[82] C. Kaiser and B. Herbst, "Smart Engineering for Smart Factories : How OSLC Could Enable Plug & Play Tool Integration," in *Mensch und Computer 2015 Workshopband*, 2015, pp. 269–278.

[83] C. E. Salloum, "Seamless Integration of Test Information Management and Calibration Data Management in the Overall Automotive Development Process," 2015.

[84] R. Bogusch, S. Ehrich, R. Scherer, T. Sorg, and W. Robert, "A Lean Systems Engineering Approach for the Development of Safety-critical Avionic Systems," in *Proceedings of the 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, Toulouse, France, 2016, p. 11.

[85] J. Munir, "Information Integration using a Linked Data approach ," 2015.

[86] M. Elaasar and Adam Neal, "Integrating Modeling Tools in the Development Lifecycle with OSLC: A Case Study," *Model-Driven Engineering Languages and Systems*, vol. NA, pp. 154–169, 2013.

[87] T. Bosch and K. Eckert, "Requirements on RDF Constraint Formulation and Validation," in *Conf. on Dublin Core and Metadata Applications*, 2014, pp. 95–108.

[88] T. Bosch, A. Nolle, E. Acar, and K. Eckert, "RDF Validation Requirements - Evaluation and Logical Underpinning," *arXiv preprint arXiv:1501.03933*, 2015.

[89] V. Choi, "Big Metadata : A study of Resource Description Framework (RDF) technologies to enable machine-interpretable metadata in biomedical science," Ph.D. dissertation, Stockholm University, 2015.

[90] I. Boneva, "Comparative expressiveness of ShEx and SHACL (Early working draft)," *hal-01288285*, 2016.

[91] B. Gallina, "A model-driven safety certification method for process compliance," *Proceedings - IEEE 25th International Symposium on Software Reliability Engineering Workshops, ISSREW 2014*, pp. 204–209, 2014.

[92] J. Conallen, "OSLC Architecture Management Specification Version 2.0," 2011. [Online]. Available: http://open-services.net/wiki/architecture-management/OSLC-Architecture-Management-Specification-Version-2.0/

[93] "Web Ontology Language (OWL)," 2012. [Online]. Available: https://www.w3.org/2001/sw/wiki/OWL

[94] H. Pérez-Urbina, E. Sirin, and K. Clark, "Validating RDF with OWL Integrity Constraints," 2012. [Online]. Available: http://docs.stardog.com/icv/icv-specification.html

[95] P. F. Patel-Schneider, "Using Description Logics for RDF Constraint Checking and Closed-World Recognition," *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, vol. abs/1411.4, pp. 247–253, 2014.

[96] S. Staworko, I. Boneva, J. E. Labra Gayo, E. G. Prud, and H. Solbrig, "Complexity and Expressiveness of ShEx for RDF," vol. i, pp. 1–17, 2014.

[97] B. Gallina, A. Gallucci, K. Lundqvist, and M. Nyberg, "VROOM & cC: a Method to Build Safety Cases for ISO 26262-compliant Product Lines," *SAFECOMP Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems (SASSUR)*, pp. 1–12, 2013.

[98] L. Bass, P. Clements, and R. Kazman, *Software Architecture and practice*, 3rd ed. Addison-Wesley, 2013.

[99] B. Gallina, K. Padira, and M. Nyberg, "Towards an ISO 26262-compliant OSLC-based Tool Chain Enabling Continuous Self-assessment," *10th International Conference on the Quality of Information and Communications Technology- Track: Quality Aspects in Safety Critical Systems*, 2016.

[100] I. Sljivo, B. Gallina, J. Carlson, and H. Hansson, "Generation of safety case argument-fragments from safety contracts," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8666 LNCS, pp. 170–185, 2014.