Development and evaluation of a framework for semi-automated formalization of automotive requirements

Ariel Syrko

This thesis is presented as part of Degree of Master of Science in Electrical Engineering

Blekinge Institute of Technology
June 2015
This thesis is submitted to the Department of Applied signal processing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Electrical Engineering with Emphasis on Signal Processing. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author:
Ariel Syrko
Blekinge Institute of Technology
Address: SE – 371 79 Karlskrona, Sweden
E-mail: ari.syrko@gmail.com

Advisors:
Prof Wlodek J. Kulesza
Department of Applied signal processing, Blekinge Institute of Technology
Address: SE – 371 79 Karlskrona, Sweden
Phone: +46 455 385898
Email: wlodek.kulesza@bth.se

Dr Guillermo Rodriguez-Navas
Mälardalen University
School of Innovation, Design and Engineering
Tel: +46-(0)21 101 356
Email: guillermo.rodriguez-navas@mdh.se

Jonas Westman
Scania CV AB/Royal Institute of Technology
Email: jowestm@kth.se
Quantity and intricacy of features implemented in vehicle have expanded rapidly over a past few years. Currently vision of autonomous vehicle is no longer a dream or SF movie, but instead a coming reality. In order to reach the better quality and high safety, advanced verification techniques are required. Simulink Design Verifier is a model checking tool based on formal verification, which can be effectively used to solve problems concerning error detection and testing at earlier stages of project. The transformation of requirements written in traditional form into Simulink Design Verifier objectives can be time consuming as well as requiring knowledge of system model and the verification tools.

In order to reduce time consumption and to guide a user through the system model and the verification tool, the semi-automated framework has been developed. An implementation of restricted English grammar patterns into Simulink objects supports description of patterns to engineers and reduces time consumption. The developed framework is flexible and intuitive hence can be a solution for other branches of industry, but further tests and verification would be required.

This thesis highlights the whole process of transformation system requirements written in natural language into Simulink Design Verifier objectives. The Fuel Level Display System model currently used by almost all Scania’s vehicles is analysed. Limitations and errors encountered during development process like a flexibility of Simulink Design Verifier to capture requirements and the patterns behaviour or ambiguity of system requirements are analysed and described in this thesis.
ACKNOWLEDGEMENT

First of all, I would like to thank my external supervisor, Guillermo Rodriguez-Navas, for giving me the great possibility to perform this master thesis inside the company, such as Scania and for his suggestions and support.

Special thanks to my industrial supervisor, Jonas Westman, always trying to explain everything really clear, for his constructive feedback and support during entire thesis period. I could not find a better supervisor.

I would like to thank my university supervisor, Wlodek Kulesza, for giving me valuable tips about the Master Thesis, and for taking time to review my work and sharing his views about my project work.

I thank Predrag Filipovsikj and Mattias Nyberg for their great suggestions and advices during the entire thesis period.

Last but not least, I would like to thank all wonderful people that I have met during this amazing experience, without them it would not be the same.
# Table of Contents

Abstract .................................................................................................................................... iii  
Acknowledgement ................................................................................................................... iv  
Table of Contents ..................................................................................................................... v  
List of abbreviations ............................................................................................................... vii  
List of figures ......................................................................................................................... viii  

## 1 Introduction ....................................................................................................................... 1  
      1.1 Problem statement and goals ....................................................................................... 3  
      1.2 Related work ................................................................................................................ 4  
      1.3 Research environment ................................................................................................. 5  

## 2 Background ....................................................................................................................... 7  
      2.1 Electrical system in Scania .......................................................................................... 7  
      2.2 Simulink, Simulink Design Verifier ............................................................................ 8  
          2.2.1 Simulink .................................................................................................................. 8  
          2.2.2 Simulink Design Verifier overview ......................................................................... 8  
          2.2.3 Formal methods in Simulink Design Verifier .......................................................... 9  
          2.2.4 Error detection ...................................................................................................... 10  
          2.2.5 Detecting division by zero and integer overflow ...................................................... 10  
          2.2.6 Detecting dead logic ............................................................................................. 10  
          2.2.7 Detecting assertion violations ................................................................................ 11  
          2.2.8 Verification of requirements according to designs .................................................. 11  
          2.2.9 Model Coverage Analysis ..................................................................................... 12  
      2.3 Specification pattern system ...................................................................................... 13  

## 3 Fuel Level Display System ............................................................................................. 16  
      3.1 Fuel Level Display System ....................................................................................... 16  
      3.2 Model of Fuel Level Display System ......................................................................... 17  

## 4 Methods and results ........................................................................................................ 22  
      4.1 Manual transformation of requirements into patterns ............................................... 22  
      4.2 Mapping requirement signals with Fuel Level Display System model signals ....... 27
4.3 Manual transformation of requirements directly into Simulink Design Verifier objects 27
4.4 Manual transformation of the patterns into Simulink Design Verifier objects ........ 35
4.4.1 Verification of results ............................................................................................... 42

5 Study findings.............................................................................................................. 44
5.1 Transformation of requirements into patterns ....................................................... 44
5.2 Direct transformation of informal requirements into Simulink Design Verifier objectives ...................................................................................................................... 45
5.3 Discussion.................................................................................................................. 46

6 Conclusions................................................................................................................ 48
6.1 Summary................................................................................................................. 48
6.2 Future work............................................................................................................. 48

References ................................................................................................................. 49
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COO</td>
<td>Computational Tree Logic</td>
</tr>
<tr>
<td>CTL</td>
<td>Coordinator</td>
</tr>
<tr>
<td>EMS</td>
<td>Engine Management System</td>
</tr>
<tr>
<td>FLDS</td>
<td>Fuel Level Display System</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic Control Unit</td>
</tr>
<tr>
<td>GIL</td>
<td>Graphic Interval Logic</td>
</tr>
<tr>
<td>ICS</td>
<td>Instrument Cluster System</td>
</tr>
<tr>
<td>LTL</td>
<td>Linear-time Temporal Logic</td>
</tr>
<tr>
<td>MTL</td>
<td>Metric Temporal Logic</td>
</tr>
<tr>
<td>QRE</td>
<td>Quantified Regular Expression</td>
</tr>
<tr>
<td>RTDB</td>
<td>Real Time Data Base</td>
</tr>
<tr>
<td>RTGIL</td>
<td>Real-Time Graphic Interval Logic</td>
</tr>
<tr>
<td>SDV</td>
<td>Simulink Design Verifier</td>
</tr>
<tr>
<td>SPS</td>
<td>Specification Pattern System</td>
</tr>
<tr>
<td>TCTL</td>
<td>Timed Computational Tree Logic</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1: Product development tool chain................................................................. 1
Figure 2: Overview of research procedure............................................................... 3
Figure 3: Electrical Control Units and priority requirements on CAN bus – general example. 7
Figure 4: Simulink Design Verifier overview diagram............................................ 9
Figure 5: Marked stateflow example................................................................. 10
Figure 6: Generated report example................................................................. 10
Figure 7: Simulink Design Verifier blocks examples............................................. 11
Figure 8: Generated test cases example............................................................. 12
Figure 9: Pattern Scopes.................................................................................. 13
Figure 10: Structured English Grammar[16].......................................................... 14
Figure 11: Fuel Display System components – variant with truck and liquid fuel engine. 16
Figure 12: The most important list of signals and constants................................... 18
Figure 13: The Fuel Level Display System model – general.................................. 18
Figure 14: The Fuel Level Display System model – more informational manner...... 19
Figure 15: The Fuel Level Display System model – division.................................. 19
Figure 16: The Fuel Level Display System model – refuel detection block............ 20
Figure 17: The Fuel Level Display System model – refuel detection block............ 20
Figure 18: The Fuel Level Display System model – refuel detection block............ 21
Figure 19: The Fuel Level Display System model – refuel detection block............ 21
Figure 20: Requirement AER201_1....................................................................... 22
Figure 21: Requirement AER201_2....................................................................... 22
Figure 22: Requirement AER201_3....................................................................... 22
Figure 23: Requirement AER201_4....................................................................... 23
Figure 24: Requirement AER201_5....................................................................... 23
Figure 25: Requirement AER201_6....................................................................... 23
Figure 26: Requirement AER201_7....................................................................... 24
Figure 27: Requirement AER201_8....................................................................... 24
Figure 28: Requirement AER201_9....................................................................... 25
Figure 29: Requirement AER201_10................................................................. 25
Figure 30: Requirement AER201_11................................................................. 25
Figure 31: Requirement AER201_12................................................................. 26
Figure 32: Requirement AER201_13................................................................. 26
Figure 33: Requirement AER201_14................................................................. 26
Figure 34: Requirement AER201_15................................................................. 27
Figure 35: Assumptions on input signals.............................................................. 28
Figure 36: Assumptions on input parameters....................................................... 29
Figure 37: Assumptions on environment signal.................................................... 29
Figure 38: Verification objectives...................................................................... 30
Figure 39: Verification objective – AER201_04.................................................... 30
Figure 40: Verification objective – AER201_05.................................................... 31
Figure 41: Verification objective – AER201_06.................................................... 32
Figure 42: Verification objective – AER201_07.................................................... 32
Figure 43: Verification objective – AER201_08.................................................... 33
Figure 44: Verification objective – AER201_09 ................................................................. 33
Figure 45: Verification objective – AER201_12/13 ............................................................... 34
Figure 46: Verification objective – AER201_14 ................................................................. 34
Figure 47: Verification objective – AER201_15 ................................................................. 35
Figure 48: General view ..................................................................................................... 36
Figure 49: Scope subsystem ................................................................................................ 36
Figure 50: Scope selection subsystem ................................................................................ 37
Figure 51: Globally, Before R, After Q, After Q until R implementation ............................. 37
Figure 52: Pattern subsystem ............................................................................................. 38
Figure 53: Pattern selection subsystem .............................................................................. 39
Figure 54: Absence, Universality, Precedence1-2, Precedence2-1 and Precedence implementation ............................................................................................................... 40
Figure 55: Max/Min duration, Bounded Recurrence, Bounded Response, Bounded Invariance implementation .................................................................................................................. 41
Figure 56: Requirement AER201_15 as test object ................................................................ 42
Figure 57: Implementation of block P, AER201_15 as test object ........................................ 42
Figure 58: Verification of AER201_15 as test object ............................................................ 43
Figure 59: Pattern frequency ............................................................................................... 44
Figure 60: Lack of information in requirements ................................................................. 45
Figure 61: Development tool-chain – discussion .................................................................. 46
1 Introduction

Over the last few years the intricacy of automotive vehicles has expanded rapidly. This is induced by significant expansion of the number of electronic, mechanical and software parts that are set inside vehicle and control a huge number of distributed purposes. Furthermore the complexity and amount of the functions implemented into vehicle rise constantly[1]. Detection of design errors at early stage of project (Figure 1) would be a great help for the engineers and huge cost reduction for Scania[2], [3]. Simulink software is the most frequently verification tool used in Scania.

System requirements written in natural text form represent behaviour that system should follow during entire developing process. Currently, simulation is the most common way of verifying the correctness of a Simulink system models. Due to the system complexity, expansion this process become convoluted and it is hard to ensure that all possible behaviour have been created and checked only by simulation[4]. Tracing back the process and documenting the outcome of the simulation likewise becomes more difficult because of complexity problems. Simulink Design Verifier provides report after verification process that explains precisely encountered errors, if any occurred. Formal verification techniques used in SDV like model checking handle intricacy issues with promising results. SDV eliminates the need to building a verification model of the system, Simulink model of the system describe real system as well as the verification model[5].

Figure 1: Product development tool chain.
Typically, engineers have to model the system then simulate it against requirements and then verify if implementation that has been done is correct. Simulink Design Verifier allow engineers to verify they work in one simple step, users can check general correctness of the implementation and verify it against requirements[5]. Possibility to follow the process in each sample time, changing input values in order to track encountered errors is irreplaceable.

Transformation of requirements written in their traditional form into Simulink Design Verifier objective require time, knowledge of the system model and experience with the verification tool[6, pp. 1–16]. SPS in form of restricted English grammar that can be used in order to reduce time consumption and to capture requirements correctly. All data should be presented in a form that can be understood by everyone and meaning of the requirement remain constant[4]. Specification Pattern System depicted in form of restricted English grammar and implemented as Simulink/Simulink Design Verifier blocks support engineers in verification process. This flexible and easy to follow approach can be probably used in all industry branches as a guide of patterns behaviour. Investigation process analyzed step by step highlighting all encountered problems and presented possible solutions can be irreplaceable during future research[7].

Entire research have been divided into two approaches, first one requires usage of Specification Pattern System during transformation in order to simplify it, and second relies on direct transformation.

The master thesis investigation goes as follows (Figure 2):

1. (a) Transformation of system requirements written in text form into structured English grammar patterns
   - Requires understanding of Specification Pattern System.
   - Requires deep knowledge of Fuel Level Display System model or absolutely correct, unambiguous, functional model requirements.

   (b) Transformation of structured English grammar patterns into Simulink Verification objects
   - Requires understanding of Specification Pattern System and Simulink Design Verifier toolbox blocks
   - Requires quite good knowledge of Fuel Level Display System

2. Transformation of system requirements written in text form directly into Simulink Verification objects
   - Require deep knowledge of Fuel Level Display System model and entire Simulink Design Verifier toolbox blocks
Current verification process in Scania is based on simulation Simulink models in order to check if they fit functional requirements, which remain quite a lot place for mistakes\[6, pp. 1–16\]. This approach with or without Specification Pattern System as a middle step is way safer and probably no longer requires person with two different types of expertise: knowledge of the system to be verified and knowledge on the formal verification tool, but could also consume a bit more time. In order to reduce time consumption, investigate limits of entire process and automation it this research was initialized. Afterwards, based on entire research process results proposed solution presented.

1.1 Problem statement and goals

Currently verification process in Scania is based usually on manual simulation of models in Simulink against requirements. Due to rapidly increasing number of features and complexity of systems Scania wants to investigate possibilities of Simulink Design Verifier and Specification Pattern System in order to minimize time, effort and resources spent on verification process.
Research questions considered in this thesis:

- What changes are needed in order to implement formal verification methods in Scania development process? Is it even possible?
  - Are the Specification Patterns flexible enough?
  - How well described system requirements are?
  - What type of errors can be encountered?
- How useful is using a restricted English grammar patterns as a middle step in entire process?
- How flexible is Simulink/Simulink Design Verifier according modelling and verification Scania’s requirements?

Research procedure goes as follows:

1. Analyse current available references to set an investigation procedure
2. Choose and analyse well-documented and non-trivial example at Scania
3. Transform requirements written in natural text form into Simulink Design Verifier objectives directly
4. Transform requirements written in natural text form into restricted English grammar patterns
5. Transform restricted English grammar patterns into Simulink/Simulink Design Verifier blocks
6. Propose a method for semi-automated verification of requirements
7. Highlight all encountered problems and propose possible solutions
8. Evaluate entire procedure according to expert knowledge and industrial usefulness

1.2 Related work

Recently, complexity expansion in automotive industry and other domains all over the world has become one of the substantial challenges. Therefore, there is quite a lot literature that involves this subject.

“Automotive behavioural requirements expressed in a specification pattern system: a case study at BOSH” [2011] by A. Post, I. Menzel, J. Hoenicke, A. Podelski investigate if specification pattern system (SPS) can be used in BOSH automotive requirements. Analyse has been done at over 289 behavioural requirements taken from automotive domain what is quite huge and convincing amount of data. The authors have shown that SPS after some evaluation can be with promising results used in automotive domain.

“Patterns in property specifications for finite-state verification.” [1999] by M. Dwyer, G. Avrunin, J. Corbett introduce specification pattern system that can be used to describe system requirements in formal form. All information in that form should be easy to understood by everyone and capture precisely requirements behaviour. Entire approach contain a mapping to various temporal logics, such as LTL, CTL.
“Real-time Specification Patterns” [2005] by S. Konrad, B. Cheng present a real time specification pattern extension to the Dwyer et al. approach. This paper also introduces restricted English grammar that express specification patterns in easy to read and understood way. Entire approach also contain extension of mapping to various temporal logics, such as LTL, CTL.

“Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar.” [2015] by M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, A. Tang present a collection of knowledge about patterns invented by Dwyer et al. and some extensions. This paper present a bit changed and extended restricted English grammar approach, even more friendly to the user then predecessor. A big advantage of this paper is a huge number of examples that explain precisely which pattern can be used under what condition.

“Reassessing the Pattern-based Approach for Formalizing Requirements in the Automotive Domain.” [2014] by P. Filipovikj, M. Nyberg, G. Rodriguez-Navas investigate possibilities of use specification pattern system (SPS) to capture Scania automotive requirements. Authors describe problems encountered during transformation and possible solutions. A big advantage of this paper is a huge number of real Scania requirements.

1.3 Research environment

Scania Group is a global company situated in Södertälje, Sweden. The company was founded in 1891 and from very beginning has focused their development efforts in heavy vehicles. Scania is present in more than 100 countries and has approximately 42,000 employees. Scania’s whole business development is based on core values like customer first, respect for the individuals and quality. Customer first works according to good knowledge of customer’s needs and contact during entire chain process, from research and development to delivery of services. Respect for the individuals means to take advantage of each employee’s knowledge, experience and ideas in order to day-to-day improvements. Quality is based on good understanding of customer’s needs throughout entire product life cycle [8], [9].

This master thesis has been done under RESA department which is focused for the research and development of high-level architecture of the electronic systems. The ‘R’ stands for “Track, cab and bus chassis development” sector, ‘E’ for “Systems development” department, ‘S’ for “System architecture and tools” section, and ‘A’ for “Systems architecture”. During work on this thesis author had access to all Scania documentation about chosen case project. Scania’s employees were very helpful and willingly answered on all questions in order to improve the output of this report.

This work was a part of the ongoing research projects VeriSpec and Espresso. The VeriSpec project goals concern development of new and adjust current verification and modelling techniques as well as development of crucial tool support to study architectural models and requirements of automotive systems. Developed solutions should be integrated with functional safety standard ISO 26262. The Espresso project works on functional system requirements, studying and improving their quality and explicitness - it is internal Scania project[10].
DISCLAIMER:

Due to the confidentiality concerns of Scania all numerical values mentioned in this master thesis are arbitrary values and do not represent the real numbers used in Scania Fuel Level Display System. Simulink model of the fuel level display system is also modified, but original behaviour has been retained.
2 Background

This chapter describes basic background theory required to understand entire research approach. For simplicity reasons description in this chapter is focused on investigated parts of presented approaches.

2.1 Electrical system in Scania

Scania’s vehicle electrical systems are mostly supported by embedded systems called Electrical Control Units (ECUs). The ECUs are responsible for supervision of the electrical systems in the vehicle, reading the sensors that are connected to them and support vehicle features. Each ECU contains a lot of features, but which features will be enabled depends on the customer’s choice when ordering. During the construction stage engineers configure vehicle and enable/disable features chosen by the customer, this phase specify the vehicle variant.

Figure 3: Electrical Control Units and priority requirements on CAN bus – general example.

Huge number of different ECUs types exists in Scania divided over tree supervision sectors of different priority (Figure 3), marked with a different colour. For example the red CAN bus presents high priority, yellow present’s medium priority and green low priority data. Each ECU is responsible for different features, so not every single ECU will be present in each vehicle. At this particular system level the Coordinator (COO) takes a role of data gateway as seen in figure 3.
2.2 Simulink, Simulink Design Verifier

2.2.1 Simulink

Simulink is a block diagram software for simulation and model-based design. It provides customizable block libraries, graphical editor and solutions for simulating and modelling dynamic systems[11].

Crucial features:

- Libraries with continuous-time and discrete-time system blocks
- Data and project management support
- Hierarchical block diagrams tools
- Variable-step and fixed-step ODE solvers
- Import C and C++ code into models
- Import MATLAB algorithms into models
- Support of model analysis tool

In Simulink it is very easy to model and then simulate a representation of real system. A huge number of blocks are available to the user in given libraries that can depict various phenomena[11].

2.2.2 Simulink Design Verifier overview

Simulink Design Verifier is a toolbox to MATLAB/Simulink software created by MathWorks. SDV uses advantages of formal verification in order to encounter and highlight difficult to localise design errors in Simulink models excluding wide range tests or simulation runs. Integer overflow, violations of design properties and assertions, dead logic and division by zero can be noticed as design errors[12]–[14].

Simulink Design Verifier allow us verify presumptions, confirm requirements at the early stage of project and without necessity to generate code. Simulation results can be used as enter data to follow encountered errors. Discrete-time part of Simulink and Stateflow usually used in embedded control systems designs is also supported by SDV[12]–[14].

Blocks in designed model with an errors or satisfied blocks are highlighted in Simulink Design Verifier letting user easily track results of verification. For each block containing error, it determines boundaries of the signals and generates a test vector that creates again the simulation error.

The most important features (Figure 4):

- Design errors detection like dead logic, division by zero, integer and fixed point overflows, and violations of design properties and assertions
- Polyspace and Prover Plug-in used as formal analysis engines
- Support of floating-point and fixed-point models
- Generation of violation examples in order to debugging and analysis support – property proving
- Test vectors can be created from functional requirements. Model coverage objectives that contain condition, decision, and modified decision/condition (MCDC)
- Support of blocks and functions for safety requirements and functional modelling

![Simulink Design Verifier overview diagram.](image)

Simulation inputs created by generated tests vectors validate functionality captured in the designed model and specified by test objectives. The design properties with the test vectors and test objectives can be exploit to verify code in processor-in-the-loop (PIL) and software-in-the-loop (SIL) test setup.

### 2.2.3 Formal methods in Simulink Design Verifier

Prover Plug-In from Prover Technology and the Polyspace formal analysis engine by MathWorks provides formal analysis methods used by Simulink Design Verifier. Formal techniques depend on mathematically demanding steps to examine through possible execution paths of the designed model for analysis instances and counterexamples. Formal analysis methods let users work with model of system behaviour in place of actual data values. System behaviour model can contain test scenarios models and verification objectives that flesh out desired and undesired behaviour of the system. Formal methods used with that kind of models complements simulation and insure proper comprehend of designed model[12]–[14].
2.2.4 Error detection

Error detection methods used in Simulink Design Verifier can uncover if exact dynamic execution scenarios exist and under what status. This knowledge can afterward be used to lead the simulation for debugging and validation or improve the designed model and its requirements.

2.2.5 Detecting division by zero and integer overflow

This part of error detection is fully automated and does not need any user contribution. After analysis user can check the outcome in an HTML report (Figure 6) or in the model. Problem solving process is simplified by supplying signals ranges on all blocks.

In the Simulink, blocks are marked as red, yellow, or green (Figure 5). Red blocks have failed the analysis, design errors detected. Test case can be generated that can create the problem case again to visualises the error. Yellow blocks means that analysis violates time limits or deciding result cannot be generated. Green blocks occur when the results are proven[12]–[14].

![Marked stateflow example.](image)

Figure 5: Marked stateflow example.

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Model Item</th>
<th>Description</th>
<th>Analysis Time (sec)</th>
<th>Test Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Overflow</td>
<td>evaluateParkingBrakeApplied</td>
<td>Overflow</td>
<td>31</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Kapitel 4. Derived Ranges

<table>
<thead>
<tr>
<th>Signal</th>
<th>Derived Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>parkingBrakeSwitchApplied 1</td>
<td>[F, T]</td>
</tr>
<tr>
<td>evaluateParkingBrakeApplied/ SFunction - output 1</td>
<td>[F, T]</td>
</tr>
</tbody>
</table>

Figure 6: Generated report example.

2.2.6 Detecting dead logic

Objects that are no longer used or verified to remain inactive throughout execution are detected as dead logic. Usually a requirement or a design error is the root cause of the problem. Dead logic guide to dead code and it is hard to notice by simulation tests. After analysis a large amount of simulations, it can be still hard to verify that an exact behaviour cannot exist[12]–[14].

10
In the report after our analysis model is marked according to the results, green colour despite active logic parts and red colour occurs if dead logic is detected.

### 2.2.7 Detecting assertion violations

Simulink Design Verifier checks if any acceptable behaviour can trigger assertions throughout the execution process. Test vector that triggers the violated assertions by acceptable scenarios can be generated and highlighted by red colour. Extra blocks for specifying constrains for the verification allow user to deep analyse designed system and encounter design defects[12]–[14].

### 2.2.8 Verification of requirements according to designs

System requirements are usually clear statements about anticipated behaviours that system can encounter and behaviours that it cannot ever encounter called safety requirements.

In order to confirm that the design behaves as requirement describe, the requirement must be first transformed from a human text language into the comprehended by formal verification engine. Simulink Design Verifier allows user to express text requirements using Simulink blocks and Stateflow diagrams. To formally verify requirement user must use verification objectives that are used to analysis if the design satisfy safety and functional requirements.

The block library provided by Simulink Design Verifier contains functions and blocks for proof objectives (Figure 7), constrains, assertions, test objectives, and a collection of temporal operators. User is able to group separate requirements and their connected verification objectives into subsystems that can be managed without design[12]–[14].

![Figure 7: Simulink Design Verifier blocks examples.](image)

Simulink Design Verifier will attempt to encounter a valid test case that fulfils the objectives. In case that objective cannot be satisfied, the design can never carry out the required specification for a granted set of constrains.
Assumption block impose signal value during verification process. User can set a range of the signal in order to reduce calculation process or impose specific behaviour, it can also reduce the verification time.

Proof objectives blocks can be used to verify propriety of design in order to satisfy the requirements. Investigate of all possible inputs that can lead to the unwanted behaviour and afterward reports on its discovery. Implemented proof objective can be verified valid or Simulink Design Verifier can create a test vector that illustrates the violation in design.

Design and requirements implemented as Simulink functions and Stateflow diagrams are able to work in parallel. The Model Coverage tool gathers data about verification objectives throughout simulation and express outcome as coverage metric in Simulink Design Verifier. In order to expedite the analysis process mark the option in proof objective about stopping the simulation when property is violated[12]–[14].

2.2.9 Model Coverage Analysis

Simulink and Stateflow implementations are analysed by Simulink Design Verifier to create test cases and limits required by industry criteria. Structural coverage parameters contain decision, condition, and altered decision/condition coverage (MC/DC).

Test generation (Figure 8) increase requirement-based analysis created by hand or gathered during simulation. Simulink Design Verifier uses current model coverage data and creates additional test vectors that satisfy coverage objectives not satisfied in requirements-based analysis phase.

Those test vectors can be used to deep analysis of absent requirements and to create better quality safety requirements. In order to generate clear and simply reports, Simulink Design Verifier diagnoses unused signals and eliminate them automatically from the test harness[12]–[14].

Figure 8: Generated test cases example.
Validation of generated test vectors can be done by Model Coverage tool that observe simulation and verify if the objectives were satisfied. Furthermore to verify objectives compatibility for decision, condition, and MC/DC compatibility, the Model Coverage tool likewise verify of proof objectives, test objectives, constrains, assumptions, lookup tables, and signal limits gathered during simulation[12]–[14].

2.3 Specification pattern system

Property specification pattern was introduced in order to simplify and divide work between design and verification engineers. Patterns were intended to catch not just a description of occurring repeatedly solutions to software issues, but in addition the requirements. Whole idea should be presented in an easy to understand form so even people new in domain can flexible use it. Practitioners can recognize similar requirements, choice pattern that coverage those requirements, and exemplify solution that fit those pattern[15].

Considering transition system that commonly have finite number of states, and a collection of transitions, perhaps tagged with events, between these states. PSP describes usually appeared requirements during the allowed state/event order in a finite-state system model. Property specification patterns express the necessary of a system’s behaviours in a range of frequent formalism[15].

Specification patterns are divided into two fundamental groups: order patterns and occurrence patterns. Possibility of mapping patterns into different formalisms, like LTL (linear-time temporal logic), CTL (computational tree logic), GIL (graphical interval logic), and QRE (quantified regular expressions) is essential. SPS is divided into two groups, even-based and state-based expressions. Pattern approach is based on capital letters like P, Q, R, S that refer for events or separation of events in event-based structure, and refer for state formulas in state-based structure[15].

![Pattern Scopes](image)

Figure 9: Pattern Scopes.
Scope (Figure 9) is a part of each pattern, which represents the range of program execution in which the pattern is obligated to hold. The scope is defined by specified a start and an end state in the pattern. For scopes with unlimited state the range in which they hold is defined at start and unlimited at the end. Generally closed-left open-right scopes are used most often, because they are simple, and safe to use[15].

Real-time specification patterns are a supplement to traditional Specification Pattern System that introduces how to specify quantitative reasoning about time. Recently in automotive domain, time bounded requirements occur quite often and they are an important part of safety requirements. Extension of mapping patterns into different formalisms like MTL (metric temporal logic), TCTL (timed computational tree logic) and RTGIL (real-time graphic interval logic) were required[16], [17].

<table>
<thead>
<tr>
<th>Start</th>
<th>1: property ::= scope &quot;&quot;,&quot; specification &quot;;&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>2: scope ::= &quot;Globally&quot;</td>
</tr>
<tr>
<td>General</td>
<td>3: specification ::= qualitativeType</td>
</tr>
<tr>
<td>Qualitative</td>
<td>4: qualitativeType ::= occurrenceCategory</td>
</tr>
<tr>
<td></td>
<td>5: occurrenceCategory ::= absencePattern</td>
</tr>
<tr>
<td></td>
<td>6: absencePattern ::= &quot;it is never the case that&quot; P &quot; holds&quot;</td>
</tr>
<tr>
<td></td>
<td>7: universalityPattern ::= &quot;it is always the case that&quot; P &quot; holds&quot;</td>
</tr>
<tr>
<td></td>
<td>8: existencePattern ::= P &quot; eventually holds&quot;</td>
</tr>
<tr>
<td></td>
<td>9: boundedExistencePattern ::= &quot;transitions to states in which&quot; P &quot; holds occur at most twice&quot;</td>
</tr>
<tr>
<td></td>
<td>10: orderCategory ::= &quot;is the case that&quot; P &quot; holds&quot; (precedencePattern</td>
</tr>
<tr>
<td>Real-time</td>
<td>11: precedencePattern ::= &quot;, then &quot; S &quot; previously held&quot;</td>
</tr>
<tr>
<td></td>
<td>12: precedenceChainPattern1-2 ::= &quot; and is succeeded by&quot; S &quot;, then &quot; T &quot; previously held&quot;</td>
</tr>
<tr>
<td></td>
<td>13: precedenceChainPattern2-1 ::= &quot;, then &quot; S &quot; previously held and was preceded by&quot; T</td>
</tr>
<tr>
<td></td>
<td>14: responsePattern ::= &quot;, then &quot; S &quot; eventually holds&quot;</td>
</tr>
<tr>
<td></td>
<td>15: responseChainPattern1-2 ::= &quot;, then &quot; S &quot; eventually holds and is succeeded by&quot; T</td>
</tr>
<tr>
<td></td>
<td>16: responseChainPattern2-1 ::= &quot; and is succeeded by&quot; S &quot;, then &quot; T &quot; eventually holds after&quot; S</td>
</tr>
<tr>
<td></td>
<td>17: constrainedChainPattern1-2 ::= &quot;, then &quot; S &quot; eventually holds and is succeeded by&quot; T &quot;, where &quot; Z &quot;. does not hold between&quot; S &quot; and &quot; T</td>
</tr>
<tr>
<td>Real-time</td>
<td>18: realtimeType ::= &quot;it is always the case that&quot; (durationCategory</td>
</tr>
<tr>
<td></td>
<td>19: durationCategory ::= &quot;once&quot; P &quot; becomes satisfied, it holds for&quot; (minDurationPattern</td>
</tr>
<tr>
<td></td>
<td>20: minDurationPattern ::= &quot;at least&quot; c &quot; time unit(s)&quot;</td>
</tr>
<tr>
<td></td>
<td>21: maxDurationPattern ::= &quot;less than&quot; c &quot; time unit(s)&quot;</td>
</tr>
<tr>
<td></td>
<td>22: periodicCategory ::= P &quot; holds&quot; boundedRecurrencePattern</td>
</tr>
<tr>
<td></td>
<td>23: boundedRecurrencePattern ::= &quot;at least every&quot; c &quot; time unit(s)&quot;</td>
</tr>
<tr>
<td></td>
<td>24: realtimeOrderCategory ::= &quot;if&quot; P &quot; holds&quot;, then &quot; S &quot; holds&quot; (boundedResponsePattern</td>
</tr>
<tr>
<td></td>
<td>25: boundedResponsePattern ::= &quot;after at most&quot; c &quot; time unit(s)&quot;</td>
</tr>
<tr>
<td></td>
<td>26: boundedInvariancePattern ::= &quot;for at least&quot; c &quot; time unit(s)&quot;</td>
</tr>
</tbody>
</table>

Figure 10: Structured English Grammar[16].

Structured English (Figure 10) grammar that supports qualitative (order and occurrence patterns), real-time and probabilistic patterns is an extension to facilitate the use of specification pattern system. It makes use of basic English phrases, but restricts their structure to phases
indicated in temporal logic. Structured English grammar helps to conciliate different ways of view: natural language and pure mathematical thought process. The grammar assist in understanding the significance of a property excluding necessary to analyse the temporal logic representation. Generally, the procedure for using structured English grammar to create a natural language structure in order: At first user choose the scope of the property (globally, before R, after Q, between Q and R, after Q until R), then pattern (occurrence or order). Afterward user can add additional requirement (time, constraint or probability)[16]–[18].

Few examples to demonstrate overall idea and understand the significance of specification pattern system:

1. *Once parking brake is applied car must stop.*

   This requirement describes cause-effect behaviour without any time boundaries and additional needs. Scope is not defined, user can suppose that this event can occur anytime during entire execution process.

   **Response Pattern:**
   
   *Globally, if (parking brake is applied) then in response (car must stop).*

2. *As shown in Fig. 3, after car is started up, graphic user panel must be synchronized within 8 seconds.*

   This requirement describes time depended behaviour, that must happened within limited time constrain. Scope is not defined, but user can suppose that this event can occur any time after mentioned state.

   **Time-Constrained Universality:**
   
   *After (car is started up), it is always the case that (graphic user panel synchronization is finished) within 8 seconds.*

3. *The refill detection shall be possible only when parking brake is applied and status signal of parking brake is valid.*

   This requirement describes double cases-effect behaviour, word *only* catches the attention on individual relation between cases and effect. Scope is not defined, user can suppose that this event can occur anytime during entire execution process.

   **Precedence Chain N1:**

   *Globally, if (refill detection is possible) then it must be the case that (parking brake is applied) and afterwards (signal status of parking brake is valid) before (refill detection was possible).*
3 Fuel Level Display System

This chapter describes quite briefly Fuel Level Display System that is used in Scania’s heavy vehicles, Simulink model of FLDS and set of functional system requirements. For simplicity reasons research led in this thesis is limited to one variant, truck with liquid fuel engine.

3.1 Fuel Level Display System

The Fuel Level Display System (FLDS) is approach used to estimate and display the fuel level to the driver, as well as to turn on alarm when fuel level in the tank is considerable low. The FLDS is used in Scania’s heavy vehicles, both in buses in trucks. Scania manufacture different versions of trucks and busses, with gas and liquid fuel engine that corresponds to different types of FLDS. Depends on the configuration like tank sizes or sensors that can be used[19], [20].

The Fuel Display System variants depend also on fuel volume, the injection system, number of fuel tanks etc. For plainness reasons only fuel type and vehicle type that have an effect on the FLDS will be considered. The reaming pieces of the vehicle will be skipped, supposing that they don’t affect the FLDS[19], [20].

The Fuel Level System main features:

- Estimate and display fuel level to the driver
- Warning the driver if fuel level is low

![Fuel Tank](image)

Figure 11: Fuel Display System components – variant with truck and liquid fuel engine.

Alarm light warns the driver of low fuel level and gauge displays current fuel level in the tank. Trucks and buses with liquid fuel engine as well as buses gas fuel engine works like presented above, but truck with gas fuel engine doesn’t use alarm light to warn user of low fuel level due to different design solution[19], [20].
The Fuel Level Display System (Figure 11) is composed by few Electronic Control Units (ECU) linked to several CAN buses. Priority of the messages sent through the CAN buses depend on the colour of the bus. For example the red CAN bus presents high priority and yellow presents medium priority data. For plainness reasons, it will be supposed that all ECUs are linked via simply one CAN bus[19], [20].

The Fuel Level Display System parts:

- Coordinator (COO) – it is the principal part, which is accountable for estimation of fuel level. In the variant for trucks with liquid fuel engine it is in addition responsible for low fuel level warning.
- Instrument cluster system (ICS) – it is accountable for correct work of gauge and low fuel level warning.
- Engine Management System (EMS) – it is accountable for fuel consumption estimation for variant with trucks and liquid fuel engine. In the variant for trucks and buses with gas fuel engine it is responsible for gathering data about fuel level in the tank.

Scania uses two main documents describing all the functional requirements connected to low fuel level alert and fuel level estimation. Functional requirements used with estimation of fuel level are presented in AE201 and functional requirements used with alert of low fuel level are depicted in AE202. These requirements are implemented inside the Coordinator (COO). All the requirements that aren’t implemented are overall system requirements or assumptions. Merely the functional requirements presented in table 1 are investigated during this research. As mentioned before only one variant is taken under investigation, truck with liquid fuel engine[19], [20].

### 3.2 Model of Fuel Level Display System

The Fuel Level Display System given by Scania is quite complex as long as our intent is to investigate all four variants. As mentioned before for plainness reasons one variant is studied in this thesis, truck with liquid fuel engine. Figure 12 presents just the most important signals and constants investigated with according to functional requirements[19], [20].

<table>
<thead>
<tr>
<th>Input Signals</th>
<th>Value</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel Rate</td>
<td>0-3212</td>
<td>L/h</td>
<td>Current fuel consumption used by engine</td>
</tr>
<tr>
<td>Sensor Fuel Level</td>
<td>0-100</td>
<td>%</td>
<td>Fuel level value estimated outside our model.</td>
</tr>
<tr>
<td>Parking Brake</td>
<td>0/1</td>
<td></td>
<td>Present if the parking brake is applied or not.</td>
</tr>
<tr>
<td>Old Fuel Volume</td>
<td>0-100</td>
<td>%</td>
<td>Fuel level saved from last shutdown.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Parameters</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel Level Total</td>
<td>10</td>
<td>Type of vehicle, truck or bus.</td>
</tr>
<tr>
<td>Fuel Level Sensor</td>
<td>15</td>
<td>Type of sensor or tank being used.</td>
</tr>
<tr>
<td>Left Tank Volume</td>
<td>13</td>
<td>Size of the left tank.</td>
</tr>
<tr>
<td>Right Tank Volume</td>
<td>13</td>
<td>Size of the right tank.</td>
</tr>
<tr>
<td>Signal Name</td>
<td>Unit</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Fuel Volume</td>
<td>m³</td>
<td>Estimated fuel level in m³.</td>
</tr>
<tr>
<td>Fuel Level</td>
<td>%</td>
<td>Estimated fuel level in %.</td>
</tr>
</tbody>
</table>

Figure 12: The most important list of signals and constants.

**DISCLAIMER:** Due to the confidentiality concerns of Scania all numerical values mentioned in this master thesis are arbitrary values and do not represent the real numbers used in Scania Fuel Level Display System. Simulink model of the fuel level display system is also modified, but original behaviour has been retained.

Figure 13 presents Fuel Level Display System model used as case study example in this master thesis.

![Fuel Level Display System model](image)

Figure 13: The Fuel Level Display System model – general.

Figure 14 present Fuel Level Display System model in more informational manner, dividing it to four sub blocks. Inputs contain information about input signals, verification of their correctness and some base calculations. Algorithm is the most important subsystem, that contains entire fuel estimation algorithms and low fuel level warnings. RouteOutputs block contain some signal calculations depended on variant used. Outputs block mostly just send the signals to real time data base (RTDB).
Figure 14: The Fuel Level Display System model – more informational manner.

Figure 15 present division on parts like mentioned before, FuelLevelEstimationAlgorithm and LowFuelLevelWarning.

Figure 15: The Fuel Level Display System model – division.
Refuel detection stateflow block (Figure 16) is one of the most important part of the system, it calculates if refill detection is possible in dependence of fuel level value and parking brake signals.

Figure 16: The Fuel Level Display System model – refuel detection block.

Figure 17: The Fuel Level Display System model – refuel detection block.

Figure 17 present a part of the refuel detection stateflow block with first assumptions and figure 18 presents the actually procedure.
Figure 18: The Fuel Level Display System model – refuel detection block.

Figure 19 presents content of fuel level estimation algorithm block. ScaleFuelConsumption block is responsible for conversion fuel consumption from litre per hour to m³ per second. GainCalculation block is responsible for calculation gain value for Kalman algorithm. FuelLevelCalculation contain Kalman algorithm and entire calculations required to estimate fuel level in the tank.

Figure 19: The Fuel Level Display System model – refuel detection block.
4 Methods and results

This chapter describes entire investigation procedure step by step and analysis of results. In order to understand problems according to transformation process it was necessary to study it during manual transformation in stages.

4.1 Manual transformation of requirements into patterns

Variant one of Fuel Level Display System, part AE201 contains totally fifteen requirements, but part of them is non-functional/non-behavioural requirements or affect another part of the system. In order to correct transformation of the requirements and clear motivation they are presented one by one[21].

Figure 20: Requirement AER201_1.

Requirement AER201_1 is non-behavioural requirement, which present just conditions according to the requirements. There is no need to transform this requirement into pattern, because it is a phenomenon requirement so it describes the system configuration or contain pure data information[18].

Figure 21: Requirement AER201_2.

Requirement AER201_2 describes signal conversion that is not a part of investigated model. There is no need to transform this requirement into pattern, can be also considered as phenomenon requirement.

Figure 22: Requirement AER201_3.
 Requirement AER201_3 describes transformation of the signal that is not a part of investigated model. There is no need to transform this requirement into pattern, can be also considered as phenomenon requirement.

![AER201_4](image)

**Figure 23: Requirement AER201_4.**

Requirement AER201_4 describes transformation of the signal according to the document [6] and values of other signals. This requirement represents an analytical expression, therefore it is out of scope of formalization.

![AER201_5](image)

**Figure 24: Requirement AER201_5.**

Requirement AER201_5 describes overall assumption on input and output signals. This requirement can be also considered as phenomenon requirement, can’t be converted into any pattern – but can be verified later as assumption.

![AER201_6](image)

**Figure 25: Requirement AER201_6.**

Requirement AER201_6 describes conditions on system upon start-up phase and has been converted into two universal patterns.

1. Globally, it is always the case that (if the state saved from last shutdown and |FLS_vol| differ more than 10% OR |FLS_vol| is above 90% then start up state shall be set to |FLS_vol|).
2. Globally, it is always the case that (if the state saved from last shutdown and |FLS_vol| differ less or are equal to 10% OR |FLS_vol| is equal or below 90% then start up state shall be set to the state saved from last shout down).
Signal names in requirement are different then signal names in the model of FLDS, that doesn’t mess up the meaning and intent of the requirement, but makes it hard to analyse for someone without knowledge of the model.

Figure 26: Requirement AER201_7.

Requirement AER201_7 contains information about refill detection signal and has been converted into universal pattern.

Globally, it is always the case that (if parkingBrakeApplied has been set continuously for 5 sec then refill detection is possible until parkingBrakeApplied become not set then refueling is considered to be finished).

Signal names in requirement are again different then signal names in the model of FLDS. This requirement describe just a part of actual requirement about refill detection, probably engineers wanted to divide it in order to describe it better, but it lost at meaning and clarity.

Figure 27: Requirement AER201_8.

Requirement AER201_8 contains information about refill detection of the tank and constrains that must be accomplished. This requirement has been converted into three universality patterns.

1. Globally, it is always the case that (if refill of the tank is done while the ECU is on then it shall be detected by the algorithm).
2. Globally, it is always the case that (the value of |FLS_%_filt| shall be stored upon fulfilment of AER201_7).
3. Globally, it is always the case that (if FLS_%_filt indicates a 30% increase from the stored value for 5 seconds or more then refill is detected).

Requirements AER201_7 and AER201_8 are probably the most ambiguous requirements in the document, both of them are connected to refill detection, but divided and written in weird way.
Requirement AER201_9 contains information about refill detection of the tank and constrains that must be accomplished. This requirement has been converted into universality pattern.

Globally, it is always the case that (if refill of the tank is detected then filter algorithm shall not be used, requirement AER201_5 does not apply and the signal totalFuelLevel shall receive its value directly from |FLS_%_filt|).

Beside wrong signal name, this requirement contains an analytical expression, so it is very difficult to formalise.

Requirement AER201_10 contain information about last value stored after the shutdown. This requirement has been converted into universal pattern.

Globally, it is always the case that (at shut down the last value of totalFuelLevel shall be stored until next start-up).

Beside wrong signal name, this requirement has been converted correctly according to pattern experts.

Requirement AER201_11 contain information about valid signal value after ECU starts. This requirement has been converted into bounded response pattern.

Globally, if (ECU starts) then in response (CMS shall send a valid value in totalFuelLevel) within 2 seconds.
Beside wrong signal name, this requirement has been converted correctly according to pattern experts.

![AER201_12]

When input signal fuelRate has status 'Error' or 'NotAvailable', the filter algorithm in AER201_5 shall continue to calculate fuelLevelTot by using only |FLS_vol| as input. K shall be changed to 5*10^(-5).

Figure 31: Requirement AER201_12.

Requirement AER201_12 contain information about status signal and constrains that must be accomplished. This requirement has been converted into universal pattern.

Globally, it is always the case that (if signal fuelRate has status 'Error' or 'NotAvailable' then filter algorithm in AER201_5 shall continue to calculate fuelLevelTot by using only |FLS_vol| as input AND K shall be changed to 5*10^(-5)).

Beside wrong signal name, this requirement contains an analytical expression, so it is very difficult to formalise.

![AER201_13]

If fuelTankSizeLeft and fuelTankSizeRight are both equal to 0, the filter algorithm in AER201_5 shall continue to calculate fuelLevelTot by using only |FLS_vol| as input. K shall be changed to 5*10^(-5).

Figure 32: Requirement AER201_13.

Requirement AER201_13 contain information about status signal and constrains that must be accomplished. This requirement has been converted into universal pattern.

Globally, it is always the case that (if fuelTankSizeRight and fuelTankSizeLeft are both equal to 0 then filter algorithm in AER201_5 shall continue to calculate fuelLevelTot by using only |FLS_vol| as input AND K shall be changed to 5*10^(-5)).

Beside wrong signal name, this requirement contains an analytical expression, so it is very difficult to formalise. Both requirements AER201_12 and AER201_13 can be merged and investigated together also.

![AER201_14]

When fuelLevelSensor has status 'Error' or 'NotAvailable' the output totalFuelLevel shall be the same. The state of 'Error' or 'NotAvailable' shall not be filtered. When the signal returns from 'Error' or 'NotAvailable' to a valid sample the filter shall be initialized with the value from the first correct sample.

Figure 33: Requirement AER201_14.

Requirement AER201_14 contain information about status signal and constrains that must be accomplished. This requirement has been converted into two universal patterns.
1. Globally, it is always the case that (if fuelLevelSensor has status ‘Error’ or ‘NotAvailable’ then totalFuelLevel shall be the same AND the state of ‘Error’ or ‘NotAvailable’ shall not be filtered).

2. Globally, it is always the case that (if the signal returns from ‘Error’ or ‘NotAvailable’ to valid sample then the filter shall be initialized with the value from the first correct sample).

Beside wrong signal name, this requirement contain high level ambiguity about “the same” statement. The same as before the state “when fuelLevelSensor has status ‘Error’ or ‘NotAvailable’” or the same as the state so ‘Error’ or ‘NotAvailable’.

Figure 34: Requirement AER201_15.

Requirement AER201_15 contain information about status signal and constrains that must be accomplished. This requirement has been converted into universal pattern.

Globally, it is always the case that (if parkingBrakeApplied has status ‘Error’ or ‘NotAvailable’ then the replacement value “Not set” shall be used).

Beside wrong signal name, this requirement has been converted correctly according to pattern experts.

4.2 Mapping requirement signals with Fuel Level Display System model signals

In order to understand the Fuel Level Display System model and connect signals from requirements to the signals that exist in the model it was necessary to create support file about names, ranges and description of the signals. Due to the confidentiality concerns of Scania the file cannot be shared in this thesis.

4.3 Manual transformation of requirements directly into Simulink Design Verifier objects

This part of investigation took the most time, because it requires deep understanding of the Fuel Level Display System model and experience in Simulink, Simulink Design Verifier. In order to verify model against chosen requirements verification environment presented at figures below has been made.
Figure 35: Assumptions on input signals.

Assumptions (Figure 35, 36 & 37) describe value or range of the signal checked during verification process. Fuel rate signal present fuel consumption by the truck and it’s calculated according to the actual fuel level in the tank. Signal fuelLevelSensor provide value from sensor in the tank, which describes fuel level. Value of oldFuelLevel has been picked randomly as a constant value because it describe last recorded sample of output totalFuelLevel. Signal pBrakeApplied present stance of parking brake, so can be either 0 or 1. Signals fuelLevelTotal and fuelLevelCachSensor are part of other variant, not investigated into this thesis. Furthermore value 0 can ensure that these signals will not affect other parts of implementation.
Figure 36: Assumptions on input parameters.

Signal parameters are constant values describing configuration and variant of FLDS. Table 1 describes the most important ones. All input signals and parameters have their own status signal that presents the current status of the signal. What is more, these signals are converted from binary code, and for example value 250 presents status GOOD, which means that the signal is valid and won't be replaced usually by 0.

Figure 37: Assumptions on environment signal.

Signal actFuelLevel is a supporting environment signal, which helps specify behaviour in the model and interdependent fuel rate and sensor value signals.
Figure 38: Verification objectives.

Figure 38 presents verification objectives investigated in this thesis. Colour orange denotes internal signals, so signals from inside the model. Therefore none of these signals can be verified in a black-box manner, digging inside the model is required to catch the meaning of requirements.

Figure 39: Verification objective – AER201_04.

This is a pure mathematical requirement (Figure 39) describing conversion from % to m$^3$, that could be even skipped during verification. Gain values are constant due to constrains in variant one that is investigated. Lack of information in requirement about status signals provides unwanted verification results – falsified. In order to satisfy this requirement, assumptions on status signals of fuel level sensor and parameters have been set to 250.
The problematic part of this requirement is statement “stable signal”, it’s difficult to define it according to investigated system. In order to verify it some percentage assumptions were made. It was assume that sensor value can contain max 4% error according to real fuel level value in the tank and then estimated fuel level, so output of investigated model can contain max 80% error according to real fuel level value in the tank. Second part of this requirement was about constant Kalman filter gain value, but lack of information about status signals and parameters made this requirement again unable to obtain satisfactory results. Figure 40 present verification part and corrections that were made in order to satisfy this requirement.
In this case again ambiguity and lack of information was encountered during verification of this requirement. The Fuel Level Display System model that was investigated doesn’t contain any signals and information about start up or shut down states, so this part of requirement was skipped in this case. According to the implementation this requirement is also dependent of refuelDetection signal, which is missing in the requirement. In order to obtain satisfying verification results, some correctness presented at figure 41 had to be done.

Simple and direct requirement (Figure 42), lack of information about status signals and because of time boundaries was quite problematic at the beginning. Due to time boundaries Simulink Design Verifier was verifying this requirement infinity amount of time, after 6 hours it was decided to try different solutions. Simplified sub-model has been done, that illustrate same behavior as original, but concentrate on this particular requirement. Time boundaries in entire
model were scaled down. After all modifications it was verified as satisfied in Simulink Design Verifier.

Time spent verifying this requirement (Figure 43) was the longest. Due to time limitation of this thesis, time boundaries and implementation based at Stateflow diagrams it was impossible to verify successfully this requirement against model. It is possible to obtain satisfactory results in SDV by developing similar stateflow diagram to the one used in the model, but it won’t be reliable result. Proposed implementation that completely illustrate requirement was still falsified by Simulink Design Verifier, probably due to different workflow in Stateflow then Simulink. A more elaborate about requirement could be a solution in this case.
This requirement (Figure 44) has been divided into two parts during verification to simplify this process, however can be done as one. Information about status signals is absent in this case also. This is one of the better requirements of FLDS, it contains a lot of information and explains exact system behaviour that should be implemented/verified except status signals mentioned above.

Figure 45: Verification objective – AER201_12/13.

These requirements (Figure 45) have been merged into one, because they are similar and dependent of same signals. In this case quite explicit information were provided except definition of ‘Error’ and ‘NotAvailable’, both statements are in the range of incorrect status signal after investigation.

Figure 46: Verification objective – AER201_14.
The most ambiguous part of this requirement (Figure 46) is “shall be the same”. It had to be done according to the model implementation in order to satisfy this requirement. After investigation it should be understand that output totalFuelLevel shall be the same as fuelLevelSensor, so shall have also status ‘Error’ or ‘NotAvailable’.

![Diagram](image)

Figure 47: Verification objective – AER201_15.

This is probably the simplest requirement (Figure 47) in entire document. Investigation in definition of status signals ‘Error’ and ‘NotAvailable’ were required. It also explains how exactly status signals works, so if status signal is incorrect that means below 242, replacement value shall be used – usually 0.

### 4.4 Manual transformation of the patterns into Simulink Design Verifier objects

Implementation of Specification Pattern System into Simulink Design Verifier is really important for engineers to understand the patterns. The engineers in Scania are usually familiar with Matlab/Simulink, most of them work with it every day so representation of the patterns in Simulink should be crucial step introduction process. The transformation should be done as clearly and precisely as possible in order to facilitate understanding it.

General view (Figure 48) of patterns in Simulink is presented at figure 49. Entire approach is divided into two branches scope and pattern. Subsystems R, Q, P, S, T represent signals used in specific pattern, events/states exactly as in Konrad et al. approach.
In subsystem scope (Figure 49) user by putting a constant value can chose one of the scopes.

Scope selection subsystem (Figure 50) contain possible scopes implementation, number in previous stage activates one case in this stage.
Figure 50: Scope selection subsystem.

Figure 51 present all implemented scopes precisely. Scope between Q and R is problematic because it requires knowledge of the future state of R. Between Q and R means that something should happen after Q and before R, but we must be sure that R will happen till the end of execution process and that is problematic. Scope after Q until R is different because it means that something should happen after Q and until R, there is no demand that R must happen in execution process.

Figure 51: Globally, Before R, After Q, After Q until R implementation.

Pattern subsystem (Figure 52) block has been done quite similar, user by putting a constant value can chose one of the pattern.
Pattern selection subsystem (Figure 53) contain possible patterns implementation, number in previous stage activates one case in this stage.
Figures 54 and 55 present implementation of the patterns in detail. The patterns that contain word “eventually” are dependent on the states that should occur till the end of the program.
execution, but there is no direct demand when. The Simulink Design Verifier isn’t able to verify such statements even if there exist a way to implement the “eventually” behavior. Due to time constrain reasons of this thesis, those patterns are skipped.

Figure 54: Absence, Universality, Precedence1-2, Precedence2-1 and Precedence implementation.
Main goal of entire approach is to present exactly same behaviour as the patterns presented in form of restricted English grammar by Konrad et al. Flexibility of this approach is quite important, so engineers can just use it immediately in their model in order to verify it. Clear division between scope and pattern, easy to follow entire procedure and simplicity are really beneficial in this case.
4.4.1 Verification of results

In order to verify this approach, few tests have been done (Figure 56). Due to time limitation of this thesis, it was impossible to completely confirm this transformation, but current results are promising.

Requirement AER201_15 were used to test this approach, the pattern implementation were combined with the Fuel Level Display System and required signals were implemented. Figure 57 present implementation of block P. Input ports are connected to Fuel Level Display System signals.

Simulink Design Verifier after 27 seconds provided results (Figure 58), tested case was satisfied. This is absolutely promising result, that point propriety of this approach. Nevertheless future tests are required in order to specify completely usefulness of this implementation.
Figure 58: Verification of AER201_15 as test object.
5 Study findings

5.1 Transformation of requirements into patterns

Figure 59 present the most frequently used patterns in the transformation.

According to the transformation and experts review, the most frequently used pattern was universality. Universality pattern is probably able to capture every behaviour, so if it seems that the requirement does not fit in any pattern, it will likely fit in universality pattern. Transformation of complex requirement that does not fit in any pattern into universality pattern create certain problem, all the complexity will be present in proposition P. The main unwritten goal is to support engineers during verification process, patterns should reduce the complexity and improve understanding. The requirements complexity presented in proposition P of universal pattern requires engineer to model this intricacy in Simulink. In the future work engineers should evaluate how useful and helpful it is.

The global scope was the only one used during transformation, it describes entire program execution. The global scope is flexible and in some sense safe, because it stays true entire execution process and everything depends on pattern. In the case of any other scope, if they become true they will stay true till the end of the execution process, so pattern is highly dependent on scope state.
During transformation of requirements into patterns it was impossible to catch all errors without verifying requirements according to the implementation. In order to ensure about the meaning of requirements, verification of results with model/requirement experts was required.

5.2 Direct transformation of informal requirements into Simulink Design Verifier objectives

Direct transformation from requirements into patterns was the most time consuming part of entire project. This was mainly due to the fact that information was missing in the requirements. In order to make sure that captured meaning of the requirement is correct it was necessary to verify it once again with system experts.

Figure 60: Lack of information in requirements.

Lack of information in requirements is factual result (Figure 60), out of the investigated case, only 24% had sufficient information to be able to do the transformation. Scania cannot take many advantage of the current requirements, as they cannot be used (as they are) for any type of formal analysis e.g. verification. Lack of universal and flexible standard e.g. “How to form system requirements properly”, practically eliminate possibility of capture everything correctly. Some requirements in investigated case were nearly impossible to understand without looking into implementation. Knowledge of implementation during verification process can affect the results. Patterns might help or not, but there is no magic solution to transform “bla bla bla” into logic text for sure. Scania has to form some framework, standard in order to be able to apply formal analysis.

Simulink/Simulink Design Verifier was flexible enough to verify almost all requirements. Common problem during verification was exceeded analysis time, in order to eliminate this
problem, more precise assumptions on the system were required. It was really helpful to set all signal ranges as precisely as possible, but also keeping in mind that setting them too strictly could change dynamics of the entire system. Time constraints in requirements also triggered exceeded verification time problem. Scale down time in entire model eliminated this issue, but in case of large amount of time constrained requirements creating sub-models could be required.

Verification of Stateflow diagrams was problematic, exploit Simulink blocks to capture Stateflow behaviour was complicated and due to time limits, it was skipped. Good quality and precise requirements could open new possibilities here, implementation of verification objectives could be easier, future work should evaluate it.

5.3 Discussion

Looking at tool-chain of entire development process (Figure 61), our current research can be affected by unclear requirements. According to experts, almost all of those unclear requirements should be represented in universal pattern with global scope. Implementation of universal pattern with global scope in Simulink for all test cases is just redundant – something should be always true during entire program execution. It will not even help engineers understand patterns, because they have to implement so many signals and behaviors within proposition P of universality pattern that will make things even more complicated.

![Figure 61: Development tool-chain – discussion.](image-url)
Once again, there is no magic solution to transform “bla bla bla” (system requirement) into logic text (pattern). Scania should first eliminate problem regarding ambiguity of requirements in order to apply formal methods. Without a standard regarding requirements different engineers in different departments even within Scania can develop requirements in different ways, so this verification approach won’t be flexible ever.

In order to make everything correctly process should go as follows: verification of model against requirements, transformation of requirements into patterns, transformation of patterns into SDV objectives and verification of model using implemented patterns. Because of unclear requirements it was necessary verify model against model and experts opinion, since using just requirements would lead to around 80% falsified cases in SDV report after verification. Good quality requirements could take less time and effort – encountered problems and errors could be different also.

Let’s assume some process simplifications:

- Requirements during development/specification phase wrote in form close to restricted English grammar form – Scania have to develop an inner standard to specify requirements. It could be more beneficial for Scania to develop standard that fit precisely they needs, since patterns were invented for software engineering, not real time systems.[15]
- Using requirements wrote in easy to follow and clear logic form in Simulink in order to verify model against requirements - redevelopment of presented implementation of patterns in SDV into new developed standard will not be a problem. It will be flexible and easy to follow.

Current solution is still a great step into formal analysis, presentation of pattern in Simulink Design Verifier can be crucial for engineers to understand how they work. The highlighted problems regarding to requirements can be used as advantage to reduce time consumption in this stage of development process. In this thesis due to time limits just one set of system requirements was investigated, and it is flatly too little.

Summing up, good quality requirements are irreplaceable in this process. If Scania can present them already in form of logic English grammar it will even reduce time consumption and can eliminate other problems.
6 Conclusions

6.1 Summary

This thesis presents the use of formal methods in model verification using Simulink Design Verifier and patterns. It shows entire verification process step by step in order to gain valuable knowledge and solutions to encountered problems. Throughout work on this master thesis it became more and more obvious that verification process within Scania using formal methods is a very cumbersome and huge topic where there is a lot future work to be done.

Introduction of formal verification methods into Scania product development process is possible. Nevertheless to obtain maximum benefits from it changes are required. Specification Patterns are one of possibilities in this case, originally invented for software engineering expose lack of flexibility in automotive domain, real time systems. However this result could be caused by unclear system requirements. According to presented results system requirements in Scania require improvement in order to apply formal methods. Without a standard regarding requirements application of formal methods in Scania will not be possible to automate entire process and make it flexible ever.

Simulink Design Verifier is great tool to verify model against requirements. It shows good flexibility in modelling and verification of Scania’s requirements. It is intuitive and easy to track, verification reports are clear and support problem solution. Time bounded requirements can be verified separately, and time should be scaled down to avoid hours or days spent on SDV verification process. Verification of stateflow diagrams was the only issue that really slow down entire procedure, however this could be caused by unclear system requirements.

Performance of patterns in Simulink Design Verifier is a wonderful support for engineers. It still requires more tests and future work, since one set of system requirements as a test case is too little. However possibility of this transformation is promising not just for Scania, but for entire industry. Possibility of creating framework that requires only implication of used signals and it is ready to start formal verification is a great result.

6.2 Future work

The approach presented in this thesis work has to be deepened and improved. Possible future works are suggested:

- Entire approach require more tests and validations, one set of system requirements cannot present whole requirement status in Scania – quality of requirements can vary a lot between projects.
- Development of standard within Scania in order to form good quality, unambiguous requirements. – using good quality requirements, encountered problems and errors can be different than these found during this thesis research.
- Include more blocks into Simulink supported by Simulink Design Verifier
REFERENCES


