Model-based Requirement Verification : A Case Study

Feng Liang[1,2] Wladimir Schamai[3] Olena Rogovchenko[1]
Sara Sadeghi[2,4] Mattias Nyberg[2] Peter Fritzson[1]
[1] PELAB - Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
[2]Scania, Sweden
[3]EADS Innovation Works, Engineering
Architecture, 21129 Hamburg, Germany
[4]School of Information and Communication Technology, KTH Royal Institute of Technology

Abstract

This paper presents a complete case study that takes a real Fuel Display System element used in Scania Trucks and applies an unified process for modelling system requirements together with the system itself and verifying these requirements in a structured manner. In order to achieve this process the system is modeled in Modelica, and requirement verification scenarios are specified in ModelicaML and verified with the vVDR (Virtual Verification of Designs against Requirements) approach.

Keywords: system modeling; requirement verification; ModelicaML

1 Introduction

As electronic systems become increasingly complex, so do the requirements that they must fulfill, both in terms of functionality and safety. Thus, maintaining the conformity between the system requirements and the system implementation manually becomes increasingly difficult and unproductive. The goal of this paper is to investigate on the basis of a real case study the integration of modeling based techniques for requirement expression with the actual implementation and the formalization of the requirement verification process.

The case study presented in this paper is a component of a Scania System Model used in real trucks. Scania is one of the leading manufacturers of heavy trucks and buses, operating in over 100 counties with over 35,000 employees and more than 110 years of history.

The Modelica language was chosen to model the system. Modelica is non-proprietary, object-oriented, equation based language for modeling multi-domain complex physical systems.

2 An Integrated Modeling Approach

2.1 Requirement Specification in the Industrial Context

The presence of Electrical and Electronic(E/E) Systems in vehicles has been increasing rapidly since the early 1970s, coming to cover a wide range of applications. Today's vehicles use around 30 Electronic Control Units (ECUs) for small cars and 80 ECUs for high-end luxury cars and this number keeps growing.

In order to simplify system representation, a concept called SESAMM (Scania Electrical System Architecture Made for Modularization and Maintenance) for SCANIA Truck and Bus electrical systems was developed [6]. However, with this approach the requirements are still kept separate from the system design.

The traditional document-based approach means that all the requirements and design information are written in document form, using natural language and graphics. Although it can be regularized, the document-based approach has fundamental limitations. Traceability and consistency are hard to ensure, since the information is spread out over different documents. Maintenance and reuse are also an issue, and since part of the documents is written in natural language, so is accuracy.

The goal of this work therefore, is to integrate the description of the system requirements into the system modeling process, thus benefitting from all the advantages of model-based ingeneering.

2.2 ModelicaML

ModelicaML [1] is an UML [2] profile and a language extension for Modelica. The main purpose of ModelicaML is to enable graphical system modeling using the standardized UML notation together with the modeling and simulation power of Modelica. ModelicaML defines different views (e.g., composition, inheritance, behavior) on system models. It is based on a subset of UML and reuses some concepts from SysML. ModelicaML is designed to generate Modelica code from graphical models. Since the ModelicaML profile is an extension of the UML meta-model it can be used as an extension of both UML and SysML. A tool suite for modeling with ModelicaML and generating Modelica code can be downloaded from [3].

2.3 Virtual Verification of Designs against Requirements(vVDR)

vVDR (Virtual Verification of Designs against Requirements) is a method that enables model-based design verification against requirements. The first version of the vVDR method and an example of its application are illustrated in [8] using ModelicaML. ModelicaML supports all Modelica constructs and, in addition, supports an adapted version of the UML state machine and activity diagrams for behavior modeling as well as UML class composition diagrams for structure modeling. This enables engineers to use the simulation power of Modelica combined with a standardized graphical notation for the creation of system models. The main vVDR method steps are:

1. **Formalize Requirements**: This step explains how to formalize requirements for design verification and how to determine which requirements can be verified using this method.
2. **Select or Create Design Model to be verified against Requirements**: This step clarifies what properties a system design model needs to have in order to be suitable for this method.
3. **Select or Create Verification Scenarios**: This step describes what the required properties of a verification scenario are.
4. **Create Verification Models**: This step explains what a verification model consists of and how it can be created.

In order to enable guidance and automation, vVDR introduces the concept of a requirement model, a design alternative model and a verification scenario. Each of these models is needed in order to create a verification model. In a scenario-based approach, a verification model will comprise one design alternative that is to be verified against a set of requirements by running one verification scenario as illustrated in Figure 1. Moreover, some additional models may be required. For example, a dedicated calculation model might be needed when the required data cannot be provided by the design model if such calculation is not part of the design.

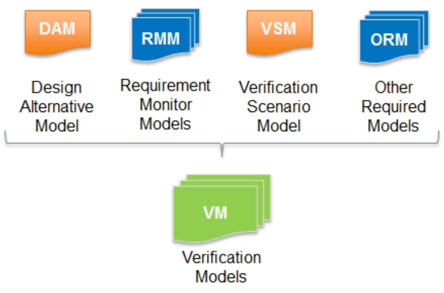


Figure 1: Different models form a Verification Model

Moreover, vVDR anticipates different roles for different tasks, that are most likely to involve different people. Each role requires specific skills and defines the responsibility for different modeling artifacts. For example, the formalization of requirements is performed by a requirements analyst. This person is in charge of requirements elicitation and negotiation. In vVDR this person is also in charge of formalizing the requirements for verification purpose because they are the most familiar with the requirements and, by formalizing them, they will reduce the probability of misinterpretation. The formalization of designs (i.e. the modeling of different design alternatives or versions) is done by the system designer, and the formalization of scenarios as well as the verification itself is done by a tester.

In vVDR a notion of clients, mediators and providers is introduced (see Figure 2). The concept is called Value Bindings [7] and allows capturing of relations that allow determining how different models should be bound when they are combined into verification models. The basic idea for the definition of bindings is the following:

- Each model that requires data from other models should express this need by creating a new mediator or by subscribing to an already existing one.
- Each mediator must have defined providers so that the correct binding code for the clients can

be derived.

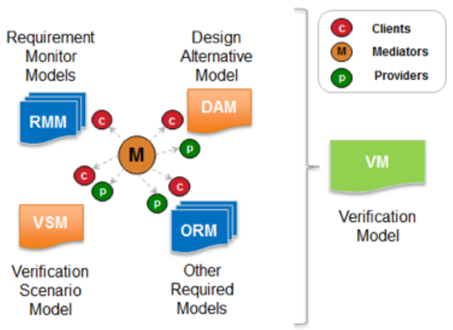


Figure 2: Clients, mediators and providers relations example

The defined relations are used to compose verification models automatically.

3 Methodology

The classification of requirements is important since it affects the requirement selection and verification process. However, there is no consensus in the field of requirement classification. The common sense divides the requirements into functional and non-functional, based on whether they answer the question of "what the system does" or that of "how the system behaves with respect to some observable attributes like performance, usability, maintainability, etc.", respectively. However, in the practice, it turns out that a more detailed classification of non-functional requirements is needed. Martin Glinz [5] proposed a taxonomy for both functional and non-functional requirements.

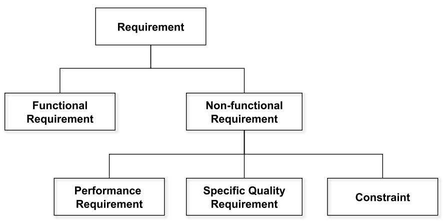


Figure 3: Requirement classification

Figure 3 illustrates the taxonomy proposed by Martin Glinz. In this view, non-functional requirements can be classified as performance requirements, specific quality requirements and constraints. Following is a definition for each category:

Function Requirement is a requirement that describes the system's reaction to input stimuli.

Performance is a requirement to specify the timing, velocity etc. inside a desired tolerance.

Specific Quality Requirement is a requirement that specifies the quality the system should have like efficiency, security, reliability, usability, maintainability etc.

Constraint is a requirement that constrains the solution space beyond what is necessary for meeting the given function, performance and specific quality requirements.

Not all the requirements are fit to be verified by the simulation model. Some require additional judgment from the stakeholder. For instance, the Specific Quality Requirement which specifies the quality of the system like reliability, maintainability etc., needs to be verified based on the experience of a stakeholder. In contrast to that, the functional and performance requirements which consist of mathematical expressions or boundaries are more suitable to be selected for the dynamic requirement verification process. This is the type of requirement we will concentrate on in this case-study.

3.1 Requirement Formalization

This is the first step of the vVDR method. The main goal of this step is to translate textual requirement statements into formal models that can be processed by computers and determine whether a particular requirement is suited to be verified with this method.

In vVDR, a requirement is formalized by first identifying the quantifiable properties mentioned in the requirement statement and then establishing the relationship between them in order to express when this requirement is evaluated and violated.

3.2 Design Model

In this step, a design model that needs to be verified against requirements is created. Since the design model will be bound with verification scenario and requirement in next steps, it should be able to provide corresponding input to requirement model. This is effectively accomplished by inspecting the mediators that indicate what data in required and of what type the values should be. When building the design model, the modeler associates the providers for each mediator.

In addition to the analysis of the mediators that need providers from the design model at hand, the designer should indicate what the potential stimuli (clients) of this system design model are, that can be set by scenarios (i.e. providers from the scenario models). In order to do so, the designer subscribes the components that are to be stimulated to existing mediators or creates new mediators respectively. The corresponding providers will be defined in verification scenarios whose creation is explained in the next section. This approach is detailed in [7].

3.3 Verification Scenario

Verification scenarios are models that capture a specific course of actions which stimulate the design model in order to cause a particular reaction. Verification scenarios are created based on requirements with the intention to verify design against requirements. One scenario can be used to verify multiple requirements and one requirement is usually verified using multiple scenarios to increase the confidence in the verification results due to the independence of the scenarios. After creating the verification scenario, it is bound with the designed system and requirements which need to be verified.

3.4 Verification Model Generation

After creating the design model, requirement model and verification scenario, this step is for binding these models in ModelicaML in order to generate executable Modelica code. By using the defined clients, mediators and providers, verification models can now be created automatically by determining valid combinations of scenarios and requirements for a selected system design model.

3.5 Requirement Verification

To express requirement violation, the attribute "status" of type Integer is used, which is created by default for each requirement. The meaning of its value is the following:

- 0 means requirement is not evaluated
- 1 means requirement is evaluated and not violated
- 2 means requirement is evaluated and violated

Now, the verification model generated from ModelicaML in the previous step can be simulated in the Modelica simulation environment. And based on the verification result, the tester will be able to analyse the system design based on the verification result.

4 A Case Study: Fuel Level Display

The case study introduced in this chapter is a Fuel Level Display System (Figure 4), used in Scania Trucks for indicating the fuel level of the truck.

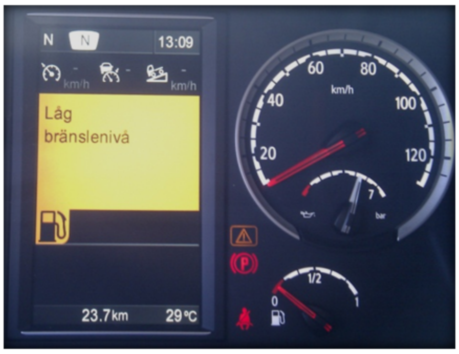


Figure 4: Dash board on Truck

The fuel level system, UF18, has two functionalities:

- fuel level estimation, which is presented as a percentage of the tank that is full. The fuel level should be displayed continuously and work for different vehicle types (truck, bus) and engine types (gas, diesel);
- fuel level warning, which is activated when the fuel level drops below a predefined value, when activated the low level fuel warning should alert the driver by some visible symbol.

These functionalities are represented by two allocation elements, AE201 and AE202 respectively.

4.1 System Architecture

The technical architecture of the Fuel Level Display System is schematized in Figure 5. Three ECUs communicate with each other through CAN-Buses. EMS (Engine Management System) sending the fuel consumption by the engine to COO (Coordinator System) which estimates the fuel level in the tank and evaluates the low fuel level warning. After processing in COO, a signal carries the estimated fuel level in the tank and the low fuel level warning to ICL (Instrument Cluster System). The gauge and bulb in ICL will indicate to the driver how much fuel is left in the tank.

4.2 Requirement Selection and Classification

In Scania, the requirements for UF18, AE201 and AE202 are described in different technical documents

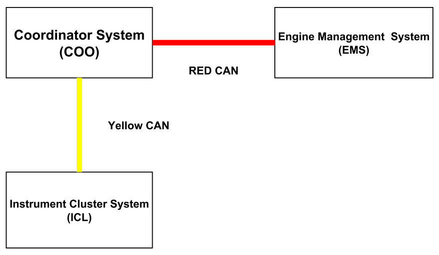

Coordinator System (COO)
Engine Management System (EMS)
RED CAN
Yellow CAN
Instrument Cluster System (ICL)

Figure 5: The technical architecture of FLD is composed of three ECUs : Coordinator ECU(COO), Engine Management System(EMS) and Instrument Cluster System(ICL).

respectively. These documents are very extensive, so a subset of elements has been selected for modeling in this case study [4].

$UFR18\_1$	The indicated fuel level shall not deviate more than $\pm 5\%$ from the actual volume in the tank.
$UFR18\_4$	The low fuel level warning shall warn one time when the estimated fuel level reaches below a limit of the measurable volume in the tank. The limit should be 10% for tank sizes below and equal to 900 Liters and 7% for larger tanks.

Based on the requirement classification presented in Section 1, $UFR18\_1$ is a performance requirement that specifies the tolerance of the indicated fuel level. The other requirement $UFR18\_4$ is a functional requirement describes how the low fuel level warning behaves with respect to the estimated fuel level.

4.3 Requirement formalization

The next step is to formalize the following requirement statement "$UFR18\_1$: The indicated fuel volume shall not deviate more than $\pm 5\%$ from the actual volume in the tank." The quantifiable properties are the:

- **Indicated fuel volume** (of type Real)
- **Actual volume in tank** (of type Real)
- And the **tolerance** of $\pm\%$ (constant of type Real and the value 0.05)

Note, that there is no precondition that defines when this requirement is valid, i.e., this requirement shall not be violated at any time. A possible precondition could be that this requirement is only valid as long as the truck is on. In this case the additional quantifiable property identified would be the fact that the truck is on, i.e. "truck is on (of type Boolean)".

Since this requirement should be checked at all times we only need to express when it is violated or not violated as follows:

```
status = if abs(indicatedFuelLevel
                - actualVolumeInTank) >
          actualVolumeInTank * tolerance
         then 2 else 1
```

The code sets the attribute `status` to 2 (i.e. evaluated and violated) or 1 (evaluated and not violated) depending on whether the absolute value of the difference between the indicated fuel level and actual fuel level in tank is greater than the allowed tolerance or not.

Consider another requirement statement: *"$UFR18\_4$: The low fuel level warning shall warn one time when the estimated fuel level reaches below a limit of the measurable volume in the tank. The limit should be 10% for tank sizes below and equal to 900 liters and 7% for larger tanks."* The quantifiable properties that are mentioned in this statement are:

- **Estimated fuel level** (of type Real)
- **Warning active** (of type Boolean)
- **Limit** (constant of type Real)
- **Size of the tank** (constant of type Real)

Again, there is no precondition for this requirement so it shall not be violated at any time. To express the violation we could define the status to be:

```
status = if (estimatedFuelLevel
              < sizeOfTank * limit)
         and not warningActive
         then 2 else 1
```

All identified properties are inputs that are to be set to the corresponding data from other models, for example the design model or models that capture the design parameters.

4.4 Design Model

In this section, a design model is written in Modelica. Figure 6 illustrates the breakdown of the fuel level display system. It consists of four levels from

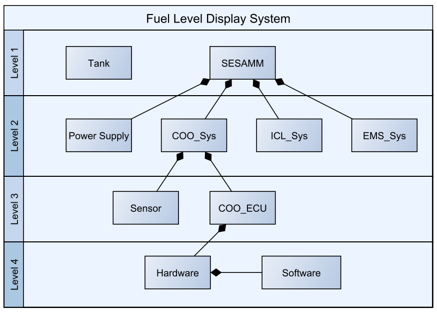


Figure 6: Breakdown of System Design

SESAMM to hardware and software. In the software domain, the application software is implemented in C code generated from the Simulink model through Real-time Workshop(RTW).

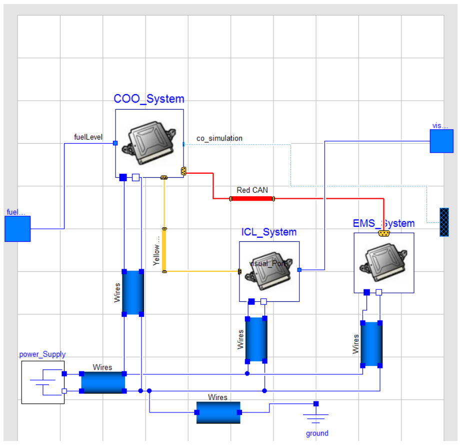


Figure 7: Second level of the system

Figure 7 shows the class diagram of the second level, different ECUs connecting with each other through different CAN-Buses. SESAMM uses different colors of CAN-Buses in order to distinguish between the most safety crucial ECUs and the less safety crucial ECUs. Furthermore, the port located on the top-right of the model carries the `indicatedFuelLevel` and `warningActive` calculated by the Simulink model.

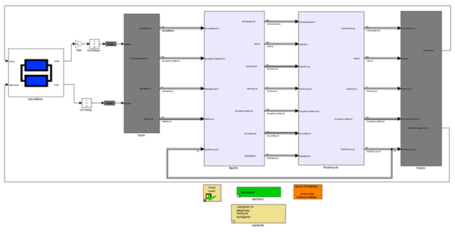


Figure 8: Joint Simulation in Dymola and Simulink

The Design model and the Simulink model are simulated through a built-in Dymola-Simulink interface as shown in Figure 8. The interface provides the Simulink model with two inputs, fuel rate from the Engine Management System and the fuel level which is measured by a sensor.

4.4.1 Verification Scenario

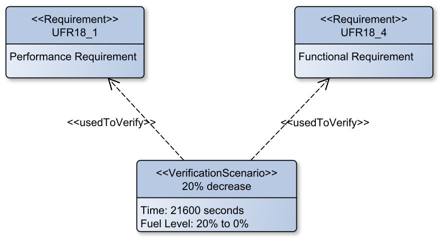


Figure 9: Verification Scenario

After having designed the system, the next step is to create a verification scenario (Figure 9) in order to verify whether the designed system fulfills the requirements. For the fuel level display system, the verification scenario describes how the fuel level in the tank decreases with respect to time. In addition, by inspecting the mediators that represent the need for simulation of the design models, the tester will define corresponding providers that are associated with the mediators.

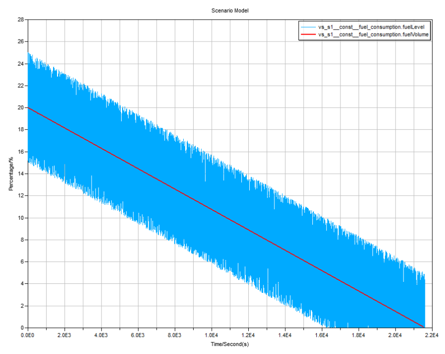
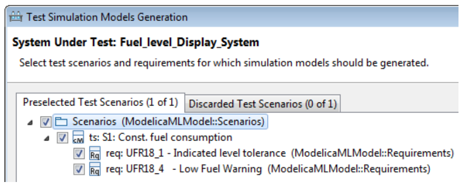


Figure 10: Simulation Result of Scenario Model

In this case study, a verification scenario describes the fuel level in the tank decreasing from 20% to 0% of the capacity of the tank. The verification scenario provides two inputs to the design model, Fuel level and Fuel Volume. Fuel level represents the fuel level measured by the sensor which consists of a noise signal caused by the shaking of tank during driving. The fuel volume represents the ideal fuel volume in the tank. In Figure 10, the blue line represents the fuel level and the red line represents fuel volume. By using this verification scenario, $UFR18\_1$ and $UFR18\_4$ can be verified at the same time.

4.5 Verification Model Generation



Test Simulation Models Generation

System Under Test: Fuel_level_Display_System

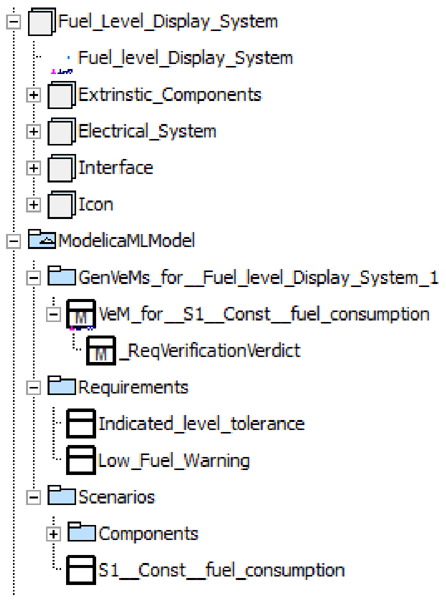
Select test scenarios and requirements for which simulation models should be generated.

Preselected Test Scenarios (1 of 1) | Discarded Test Scenarios (0 of 1)

- ☑ Scenarios (ModelicaMLModel::Scenarios)
  - ☑ ts: S1: Const. fuel consumption
    - ☑ req: UFR18_1 - Indicated level tolerance (ModelicaMLModel::Requirements)
    - ☑ req: UFR18_4 - Low Fuel Warning (ModelicaMLModel::Requirements)

Figure 11: Verification Models Generation

In ModelicaML, the verification model can be generated by binding the design model, to the verification scenario and the requirements. By using the defined clients, mediators and provider verification models can now be created automatically by determining valid combinations of scenarios and requirements for a selected system design alternative model [7] as illustrated in Figure 11. The generated verification models comprise the components that are bound correctly and are ready to be simulated.



```
Fuel_Level_Display_System
    Fuel_level_Display_System
    Extrinstic_Components
    Electrical_System
    Interface
    Icon
ModelicaMLModel
    GenVeMs_for__Fuel_level_Display_System_1
        VeM_for__S1__Const__fuel_consumption
            _ReqVerificationVerdict
    Requirements
        Indicated_level_tolerance
        Low_Fuel_Warning
    Scenarios
        Components
        S1__Const__fuel_consumption
```

Figure 12: Verification Model in Modelica Simulation Environment

Figure 12 shows the package when the importing verification model to the Modelica simulation environment. The package ModelicaMLModel was created in ModelicaML. It consists of a Verification Model, a Scenario Model and a Requirement Model. The verification model binds other models together and simulates the results.

4.6 Requirement Verification

The verification result of requirement $UFR18\_1$ is shown in Figure 13 and the verification result of requirement $UFR18\_4$ is illustrated in Figure 14. As mentioned previously, there is no precondition for these two requirements, so they should be evaluated during the whole verification process.

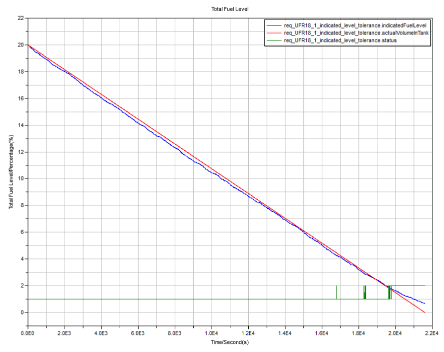


Figure 13: Verification Result of $UFR18\_1$

Figure 13 shows the verification result of the Total Fuel Level element. The red line represents the actual volume in the tank and it decreases progressively from 20% to 0%. The blue line shows the indicated fuel level from the Instrument Cluster System. Finally, the green line shows the requirement status. The status starts at 1 which means the requirement is evaluated and not violated until around 20000 seconds. From around 20000 seconds, the status changes to 2 which means that the requirement is evaluated and violated. So the corresponding requirement $UFR18\_1$ is fulfilled in the first 20000 seconds, then it violated until the end of the simulation.

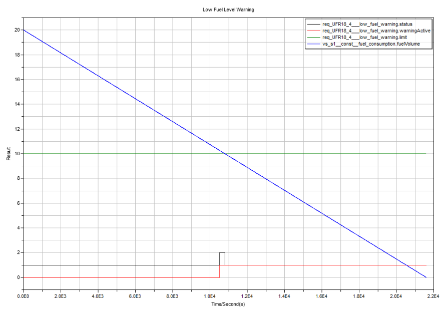


Figure 14: Verification Result of $UFR18\_4$

Figure 14 shows the verification results of the Low Fuel Level Warning element. The blue line shows the indicated fuel level from Instrument Cluster System which decreases progressively from 20% to around 0%. The green line shows the threshold at which the low fuel level warning must be enabled. The black status line illustrates that the requirementUFR18_2 is violated during 10541 second to 10776 second.

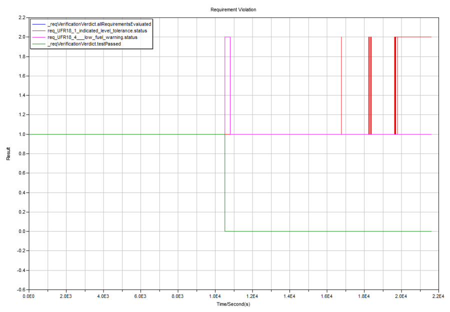


Figure 15: Requirement Violation

Figure 15 is the monitor that shows whether the test is passed or not. As we can see from picture, the is test passed in the first 10541 seconds since both requirements are not violated. After 10541 seconds, requirement UFR18_4 is violated which means that the test fails.

5 Conclusion and Future Work

This case study illustrates the approach to formalizing requirements from document-based format through the vVDR methodology, and generating verification scenarios to test whether the system fulfills these requirements.

The reasons for choosing vVDR approach are its requirements formalization approach, its scalability and the level of possible automation. The way requirements are formalized detects inconsistencies or incompleteness of requirements, it allows expressing requirements monitors using the same formalisms that are used to formalize designs or scenarios, and it allows determining which requirements can be verified using simulations. This is possible based on the knowledge which design models are or will be in place. The generation of verification models, provided by the vVDR approach and its implementation in ModelicaML, automates the process of solving the combinatorial task to select scenarios that are appropriate to stimulated a given design alternative model and all requirements that can be verified by running this scenario. For a small number of requirements, scenarios and design alternatives this approach may be overdone. However, assuming a large number of these artifacts in a real-life project the provided automation is expected to significantly improve the process efficiency.

The goal is to further investigate and generalize the modelling methodology in the industrial context, by applying to to larger test cases and formalizing the process. This work is part of a larger project on a integrated toolchain from documentation formalization through to requirement verification and fault tolerance analysis.

6 Acknowledgement

This work has been supported by Scania, by the ELLIIT project, the Swedish Strategic Research Foundation in the EDOp and HIPo projects, and Vinnova in the RTSIM and ITEA2 OPENPROD projects. The Open Source Modelica Consortium supports the OpenModelica work.

References

[1] Modelica Association: Modelica: A Unified Object- Oriented Language for Physical Systems Modeling: Language Specification Version 3.2. http://www.modelica.org.

[2] Object Management Group: OMG Unified Modeling Language TM (OMG UML). http://www.uml.org/.

[3] OpenModelica Project: ModelicaML - A UML Profile for Modelica. http://www.openmodelica.org/modelicaml.

[4] C. Erlandsson. Revising the user function requirements document of fuel level display for compliance with iso 26262 and literature. Master's thesis, 2012.

[5] Martin Glinz. On non-functional requirements. In *Requirements Engineering Conference*, pages 21–26, 2007.

[6] L. Hans. The SESAMM concept. PD1422538, 2003.

[7] Wladimir Schamai, Peter Fritzson, Christiaan J. J. Paredis, and Philipp Helle. Modelicaml value bindings for automated model composition. In *Proc. of Symposium on Theory of Modeling and Simulation (TMS/DEVS 2012)*, 2012.

[8] Wladimir Schamai, Philipp Helle, Peter Fritzson, and Christiaan J. J. Paredis. Virtual verification of system designs against system requirements. In *MoDELS Workshops*, pages 75–89, 2010.