Chapter 4
Semi-formal Representation of Requirements for Automotive Solutions Using SysML

Liana Muşat, Markus Hübl, Andi Buzo, Georg Pelz, Susanne Kandl, and Peter Puschner

Abstract As system complexities are growing with increasing numbers of requirements, the difficulties to manage, process and verify natural language requirements and to keep quality are also increasing. In safety-related applications, as in the automotive domain, this necessity is more pronounced because of the regulations and standards imposed by authorities. Semi-formal representation of requirements is an approach that helps making them more understandable and rigorous.

This chapter deals with semi-formal representation using SysML of two automotive analogue-mixed signal systems, an electronic power switch and an airbag safety circuit. We use diagram-based modelling in order to represent requirements, structure and behaviour, enabling the linking different elements that define the composition and the functionalities of the desired product. We focus on the particular behaviour of such devices and the continuous quantities related to them with emphasis on the two real scenarios.

L. Muşat (✉)
Automotive Power, Infineon Technologies AG, Villach, Austria

Institute of Computer Engineering,Vienna University of Technology, Vienna, Austria
e-mail: Liana.Musat@infineon.com

M. Hübl
Automotive Power, Infineon Technologies AG, Villach, Austria
e-mail: Markus.Huebl@infineon.com

A. Buzo
Methodology development - Automotive, Infineon Technologies Romania&Co SCS, Bucharest, Romania
e-mail: Andi.Buzo@infineon.com

G. Pelz
Methodology development - Automotive, Infineon Technologies AG, Neubiberg, Germany
e-mail: Georg.Pelz@infineon.com

S. Kandl • P. Puschner
Institute of Computer Engineering, Vienna University of Technology, Wien, Austria
e-mail: susanne@vmars.tuwien.ac.at; peter@vmars.tuwien.ac.at

F. Oppenheimer, J.L. Medina Pasaje (eds.), *Languages, Design Methods, and Tools for Electronic System Design*, Lecture Notes in Electrical Engineering 361,
DOI 10.1007/978-3-319-24457-0_4

4.1 Introduction

Over the last decades the number of electrical and electronic (E/E) systems in a car has grown substantially and the trend will be maintained for the next years to satisfy the demand of the market and users for increased automatic control. In parallel, the development complexity and the integration difficulty of E/E systems have also increased due to the high number of requirements that define the automotive sub-systems. Within this context, the automotive community is facing several challenges. First, the management of requirements, i.e. documenting, analysing and tracing of requirements, has become very difficult. Second, there is no explicit association between the description of the requirements in natural language provided by stakeholders and their implementation.

The direct implementation of natural language requirements could raise misunderstanding and misinterpretation which could lead to problems in a later phase of the development project which is considerably more costly than at the beginning. Typical problems are omission of the required features, over-specification or implementation of redundant features caused by a misunderstanding or a lack of completeness and consistency of requirements.

Third, most typical E/E automotive implementations contain elements from the analogue-mixed signal (AMS) domain. While it is relatively straight forward to describe local functionality with natural language, it appears to be more difficult to structure and describe functional relations between various internal and external components. In many existing specification interdependencies between separate blocks are often spread across the document.

In addition, the newly released ISO 26262 standard provides regulations for the development process of automotive systems in order to handle safety-related requirements systematically. The main objective of the standard is to increase the safety by reducing risk of malfunctions that would cause harm to people. This is managed by addressing the whole product lifecycle that supports tailoring of the necessary activities during the lifecycle phases (management, development, production, operation, service and decommission).

Nevertheless, the mentioned challenges can be tackled by more robust requirements through additional formality. The degree of the formalization is debatable though. Fully formalized requirements assure a full definition of the component or the system, becoming objectives for implementation as well as for automatic verification. On the other hand, a formal representation is very complex and requires a deep knowledge of domain, language and tools. Hence the number of developers is restricted to a few experts which are able to deal with the formal requirements.

Semi-formal representation is a reasonable trade-off for adding formality to the requirements without losing the simplicity of the representation. Moreover, it can make the formulation and the presentation of the requirements more intuitive.

One example of standardized semi-formal graphical languages is the System Modelling Language (SysML) [1]. It was developed by the International Council on Systems Engineering's (INCOSE [2]) Model-Driven Systems Design workgroup.

It is dedicated to systems engineering applications (including both hardware and software). SysML is based on the software modelling language UML (Unified Modelling Language), extending the capabilities of UML for the system domain, and excluding specific software parts. The systems' architecture, its behaviour, the requirements and the relationships between all elements of these aspects can be described in SysML. Although semi-formal representations of requirements and SysML in particular are widely used in the software domain, there are fewer attempts in utilizing these for hardware systems or mixed signal components. One explanation for that could be that hardware manufacturers inherited the tradition to describe the specification mainly in natural language and the higher difficulty in representing such systems, especially when it comes to the AMS domain, as most authors for semi-formal modelling in literature make full abstraction of the analogue functionality.

In this work, we present the advantages of a model-based representation of two AMS components from an automotive application. We use SysML to organize the requirements, to specify the interaction of the components with the environment and other components. This study also shows a description of the high-level structure and the behaviour of the components in SysML. The emphasis is put on illustrating the safety mechanisms while there is the possibility for further extension to all requirements. We demonstrate that AMS functionalities and quantities can be successfully modelled with SysML and that such modelling helps in overcoming the above-mentioned issues.

4.2 Related Work

Semi-formal languages are characterized by a well-defined syntax but without unambiguous and precise semantics. They are a good compromise between full formal representations, which can be simulated in a deterministic way, and natural language representations which are vague and tend to be widely misinterpreted in the engineering field due to the diverse backgrounds and experiences among the members of a development team.

Formal methods are mathematics, logic or algebra-based languages used for specification and verification, with well-defined syntax, semantics and rules. They are most popular in the software field. The weaknesses of formal methods are the limits of computational models, the high initial cost for initial implementation and most importantly the usability [3, 4].

On the other hand, natural language description represents the easiest way to capture and communicate the requirements. Nevertheless, with this advantage come many disadvantages for expressing requirements: lack of clarity, amalgamation, confusion and over-flexibility (the same requirement can be expressed in completely different ways).

Semi-formal methods try to inherent advantage from both approaches. They are well-structured and user-friendly; most include graphical notations, others are solely textual based like the Object Constraints Language [5].

One of the widest used and best known semi-formal notations is UML, a standardized general-purpose modelling language in the field of software systems engineering [6]. UML is an object modelling language, and thus it cannot cover all aspects of E/E system development. Architecture Analysis and Design Language (AADL) is another standardized modelling language used for model-based engineering for embedded software system architectures. AADL is a textual modelling language with graphical elements, which addresses application software runtime architecture but excludes the operational environment for the system view [7, 8].

Based on UML, SysML and AADL, an architecture description language specific for automotive embedded systems—EAST-ADL—has been developed and enhanced by several European research projects [9].

Another extension of the UML profile for model-driven development of real time and embedded applications is MARTE (Modelling and Analysis of Real-Time and Embedded systems) [10].

An extended review of several modelling languages including AADL, UML, SysML and MARTE has been conducted by Evensen and Weiss [11]. The criteria for the evaluation in the context of real-time software system applications are scope, formalism and architectural coverage. A comparison summary reveals the limitations of each language. AADL represents an abstraction of real-time operating system components without support for behaviour modelling. UML has no strict formalism and due to the support of a large number of diagrams, a consistent, semantically correct specification is practically hard to maintain. MARTE has a very complex meta-model and it suffers also from the support of large number of diagrams like UML. SysML is primarily targeted for the system engineering domain.

Other semi-formal languages are being developed in various research activities but none of them is widely spread or well-supported by tool vendors. From these only URML—Unified Requirements Modelling Language—is worth mentioning. The most interesting characteristic of URML is the traceability of requirements from different domains—functional requirements, possible threats and hazards or product features [12].

SysML attempts to overcome several of the limitations of the mentioned languages. It represents a visual modelling language that supports specification, analysis, design, verification and validation of a broad range of systems.

SysML, based on a subset of UML, was adapted for system engineering applications, extending UML's capabilities supporting continuous quantities modelling by using parametric diagrams to define performance and quantitative constraints.

This brought the advantage that this kind of representation streamlines communication between heterogeneous teams during the development of the system, as well as the communication to the stakeholders. The requirements are graphically modelled, explicitly representing the mapping and the relationships with each other.

Additionally the system decomposition can be considered in the initial phase of the development activities already.

Although the majority of literature is pointing out the application in software systems, SysML can be used to define models of different domains. Hierarchical modelling of a system and allocating the requirements to the structural and functional elements is one of the most common ways of modelling in SysML [14, 15]. These models are further used as a starting point for Hazard Analysis and FMEA [13]. Other works show the interaction of SysML with Matlab/Simulink for simulations [16]. Moreover SysML can be a base for code generation for System C and C/C++ [17–19]. Further research shows how SysML is transformed or mapped for formal verification [25, 26].

Also, SysML is widely used to support the requirements engineering process due to its specific diagram and relationships for requirements [24, 27–30].

In this research work, we go beyond the state-of-the-art by modelling the requirements for a smart automotive power switch and an airbag IC component from the AMS domain. The analogue effects are not abstracted, but instead they are detailed by the means of activity and state machines diagrams. Then, this behaviour is linked to the requirement and the structural elements. We illustrate these by giving modelling examples of the functionalities for the targeted applications.

4.3 Application To Be Modelled

The methodology was applied on two safety-relevant systems: a protected high-side switch as part of the electric control unit (ECU) that controls the ABS braking system and an IC part of the ECU of the airbag system.

4.3.1 Protected High-Side Switch Description

The hardware device whose requirements we have chosen to model with SysML is a field-effect transistor (MOSFET) equipped with protection and diagnostic functions. It is meant to work as the driver that activates the valve in an anti-lock braking system (ABS), hence the motivation for the presence of protection and diagnostic functions. It represents a typical safety-relevant automotive application which is very sensitive to safety issues. This is why the requirements for this device contain an additional set of explicit safety requirements beside the usual set of requirements. This leads to an increase of the complexity where the semi-formal representation becomes convenient. The following paragraphs give a short overview of the E/E system and its protection and diagnostic functions.

Integrated MOSFETs are widely used for various automotive switching applications, as high-side and low-side switches. High-side switches are power switches that can switch high currents into grounded loads safely. Further high-side switches

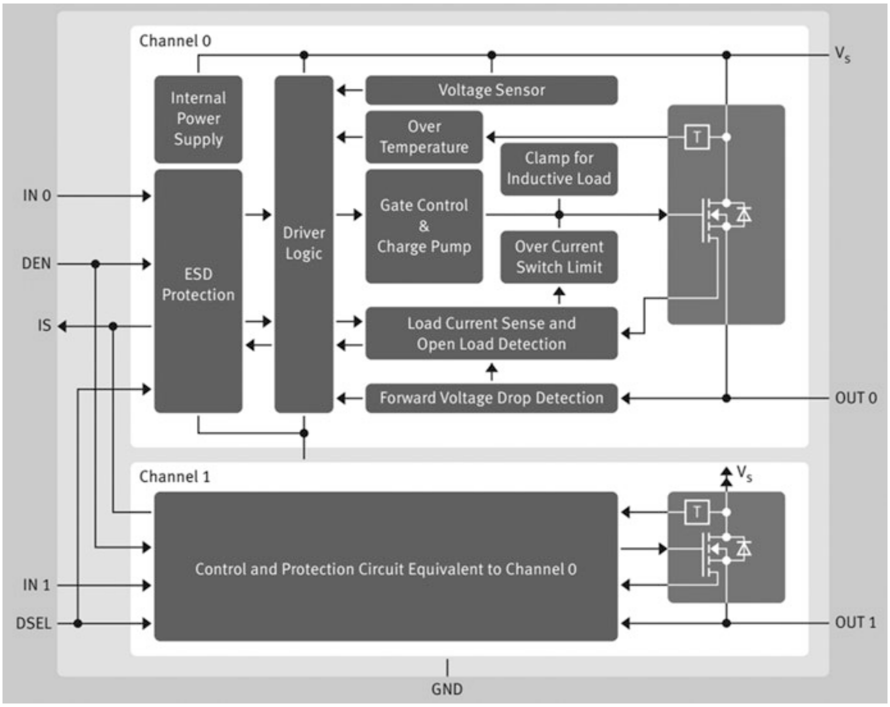
Channel 0
Internal Power Supply
ESD Protection
Driver Logic
Voltage Sensor
Over Temperature
Clamp for Inductive Load
Gate Control & Charge Pump
Over Current Switch Limit
Load Current Sense and Open Load Detection
Forward Voltage Drop Detection
T
V_S
IN 0
DEN
IS
OUT 0
Channel 1
Control and Protection Circuit Equivalent to Channel 0
IN 1
DSEL
OUT 1
GND

Fig. 4.1 Block diagram of a protected MOSFET [20]

are common in automotive applications due to less wiring and thus reduced system cost. Another advantage of a high-side configuration is the protection of the load due to the disconnection of the supply voltage in OFF state as well as the protected wiring harness [21].

Due to the high demand for safety in the automotive field, high-side switches are designed with built-in protections and diagnostic features. These protection features safeguard that the transistor will not be damaged if one or more of the operating conditions are violated or not fulfilled anymore.

An example of a block diagram for a two-channel smart high-side switch with integrated safety features is illustrated in Fig. 4.1. A large setof available protection and diagnostic features are integrated into a smart high-side switch: electrostatic discharge (ESD) protection, over-voltage and under-voltage protection, reverse battery protection, thermal shutdown protection, over-current protection, short circuit protection, loss of ground or supply voltage protection. These features guarantee that the transistor will not be destroyed or damaged in case one or more of the operating condition requirements are not fulfilled. For example, an over-voltage protection becomes active if the supply voltage is higher than the maximum operating condition and ensures that the high-side switch is not damaged in case of a load dump transient or other high voltage disturbances. In a comparable way, the under-voltage protection is activated when the supply voltage drops below the operating range. In this case the load is protected by turning off the switch to avoid unpredictable behaviour. Another safety feature is the reverse battery protection, when the supply voltage and ground connections are reversed—which keeps the integrated circuit (IC) and the load safe even when the polarity of the battery is reversed. A current limitation feature is also implemented in order to protect the IC and the load from a short-to-ground failure. The temperature monitoring logic turns off the switch in case of an over-temperature event. The shutdown logic works with a hysteresis to prevent the oscillations of the switching caused by various protections features. The protection techniques are not identical for all devices and differ from model to model and from development organization to development organization [22].

Typical automotive applications for high-side switches are:

- Resistive loads: LEDs, window heating, seat adjustments, auxiliary heating;
- Inductive loads: wipers, ABS and EBS (Electronic Braking System) valves, relays, fan motor and
- Capacitive loads: incandescent lighting and xenon lights [21].

In this application, the high-side switch is part of an ECU, together with a microcontroller (μC), which acts as the master. Some protection functions are fully integrated into the high-side switch, others work only in conjunction with external electronic components. Thus the ECU level has to be taken into consideration for handling protection requirements and functionality of the high-side switch.

The diagnostic function provides feedback to the microcontroller for normal as well as faulty behaviour. The feedback is provided as a current on a sense pin, which is proportional to the respective value on the load.

During faulty conditions the current provided by the diagnostic pin has a well-defined value or a defined range for both ON and OFF states of the transistors as well as for normal functionality and for different fault cases. In ON state, a diagnostic signal can indicate an over-temperature problem, an overload, a partial load loss or an open load. In OFF state it is important to signalize an open load or a short circuit to battery, in order to detect a fault as quickly as possible [22].

The integrated protection functions avoid a destruction of the device as well as provide protection of the connected load, while the diagnostic functions provide information to the μC about the state of the system, e.g. to support protection on the next level and to inform the driver in abnormal conditions. This makes the chosen example system an important asset for safety-related applications, e.g. for controlling an ABS application.

The ABS represents an automotive safety-relevant system used in the braking system of a car. The ECU communicates with the wheel speed sensors and based on data received, the μC controls the valves from the braking system through the high-side switch, preventing the vehicle wheels from locking up and avoiding uncontrolled skidding in critical situations [23]. Valves control represents a vital safety mechanism. A failure in the system, such as when the valve is always closed, the correct braking functionality is affected. An open valve could lead to an unavailability of the ABS system. Both faulty situations are detected by the system and signalized to the operator of the car. A faulty mechanism can lead to accidents and a loss of life or severe injuries. The protection functions of the high-side switch in the ECU system shall prevent any fault behaviour for the valve control. The diagnostic functions shall report this in order to permit the system to take the correct decision in case of a faulty mechanism [31].

4.3.1.1 Modelling in SysML

Requirements Modelling

The development of complex safety systems requires focus on requirements management from their definition at the beginning of the process until the verification and proof of implementation. The requirements are first described in natural language. This makes them susceptible to incompleteness, inconsistency or ambiguity. Using semi-formal modelling improves the quality of the requirements and of the development process by identifying the issues in the natural description at an earlier phase.

An important feature of SysML is the diagram extension for requirements. This diagram is used to specify the requirements hierarchically and their derivations as well as their relations to other model elements. One of the first steps in product or system definition is the elicitation of requirements.

This step is important for the implementation as well as for verification and validation process. The correct understanding of requirements is the key for the success of the design and the final product.

In SysML, the text-based requirements can be represented graphically in a diagram. This allows the requirements to be expressed hierarchically and linked to requirements or model elements by using different relationships (derive, satisfy, trace, verify, copy or refine). The final list of requirements is often composed of multiple internal and external sources of information. The SysML modelling capabilities offer the possibility to organize stakeholder requirements hierarchically and to group them depending on their specific type or need. These requirements can be refined to internal technical safety requirements (TSRs) by adding any necessary details further. An example of requirement organization in SysML is depicted in Fig. 4.2. It shows only a part of the requirements hierarchy. The continuous lines with crossed end represent a "nesting" relation, i.e. hierarchy relation. The dotted arrows represent the refined relation which exists between the linked requirements. The functional safety requirements (FSRs) from the client are represented in the second row, while on the third row there are represented the internal requirements—built based on the client requirements but enhanced by previous experience and knowledge about implementation. The requirement FSR01, for example—"An open load detection shall be implemented in the electronic control unit". is refined by two internal TSRs: SR_023 "The protected power switch shall provide for an open load detection in ON mode". and SR_024 "The protected power switch shall provide for

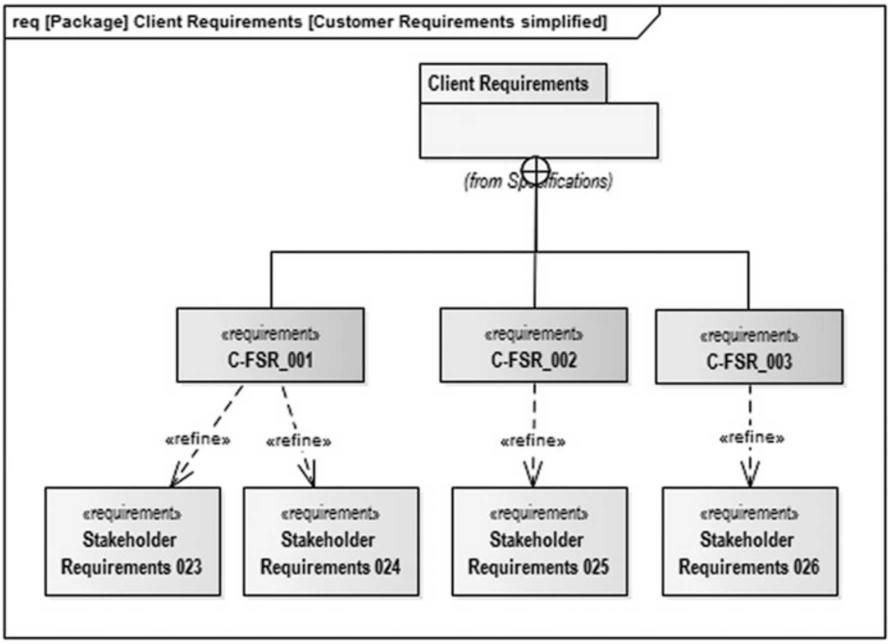
req [Package] Client Requirements [Customer Requirements simplified]
Client Requirements
(from Specifications)
«requirement» C-FSR_001
«requirement» C-FSR_002
«requirement» C-FSR_003
«refine»
«refine»
«refine»
«refine»
«requirement» Stakeholder Requirements 023
«requirement» Stakeholder Requirements 024
«requirement» Stakeholder Requirements 025
«requirement» Stakeholder Requirements 026

Fig. 4.2 Requirements organization

an open load detection in OFF mode”. Every FSR is related to a safety goal, which “Harm through a wire-break between power driver and load shall be avoided”. for the example above. This could be modelled and organized in the same way. On the other hand, the internal TSRs can be further refined into a lower abstraction level with implementation details. SR_023 comprises three internal requirements:

- “If the load current Iload is smaller than 100 uA in ON condition, open load shall be detected”.
- “Iload shall be measured through the sense current Isense at the sense pin of the power switch, which is a factor of 100 smaller than the load current Iload”. and
- “Isense itself shall be measured over the shunt resistance Rshunt using the ADC3 of the microcontroller”.

The examples above show that a high-level requirement demands the collaboration of various parts of the ECU and cannot be mapped to a single component. For example, the second internal requirement needs to be split further to become atomic and to be allocated to a component. One part of it “factor 100” is allocated to the power switch, while the other part describes the role of the sense pin, which is allocated to the power switch. It informs the user what the sense pin is supposed to do. A requirement must be split further into sub-requirements as long as it cannot be assigned to a single component (or a single function). In contrast, the third internal requirement is clear and it is allocated to the μC and not to the power switch.

As mentioned, one benefit of modelling the requirements using SysML is the structural and hierarchical organization. This can be easily done directly in SysML or in collaboration with a requirements management tool. It turned out to be especially significant when dealing with a big number of requirements. The graphical representation can be the optimal assistance in the communication with stakeholders, as it gives a clear overview of the requirements. At the same time, refining and linking the external with internal requirements is very beneficial in case of a change request from either side. This creates an easy way to identify which elements are related to the changed or deleted requirements. Additionally, the impact of the modification can be quickly evaluated and the responsible or the related persons are easily identified. Another important advantage of the SysML representation is the possibility to link the requirements with the structural and behavioural models. It offers the possibility to verify that all requirements are covered by the implementation and verification. Due to these aspects, SysML represents a very essential aspect for safety-related requirements, as it provides the proof that the requirements are proposed for implementation and are object to verification—as demanded by the ISO 26262 standard for FSR. Documents and pictures can be attached to the requirement diagrams for implementation proofing, providing a clearer understanding of the requirements depending on the tools capabilities.

Structure Definition

The structure of a system or component can be expressed using SysML block definition diagrams (BDDs) and internal block diagrams (IBDs). The BDD is used to represent the structure using blocks and the interaction between elements or with the environment. The IBD complements the aspects conveyed on BDD, by specifying the internal structure of a single block, including the connections and interfaces between the internal parts.

Figure 4.3 depicts the ECU structure with emphasis on the interaction between the μC and the protected high-side switch. The interfaces are the control input and the diagnostic feedback. The outputs of the high-side switch are used to control the respective load, i.e. the valves used for ABS. In order to achieve a clear overview of the system the battery connections as well as the ground connections are important to be depicted. This model embodies the basic view for introducing the external protection functions, for example, the output protection of the load. The same representation can be provided for all pins and the connections of the switch. Every block diagram can be further refined to an IBD. This provides a transformation from a black box view into a white box view by providing information about the internal structure of the components.

An example of the protected switch is illustrated in Fig. 4.4—it represents a SysML model of the overview depicted in Fig. 4.1. The difference between the two representations is that in SysML the blocks from a diagram are linked to blocks from other diagrams such as the requirements that define them, the system diagrams

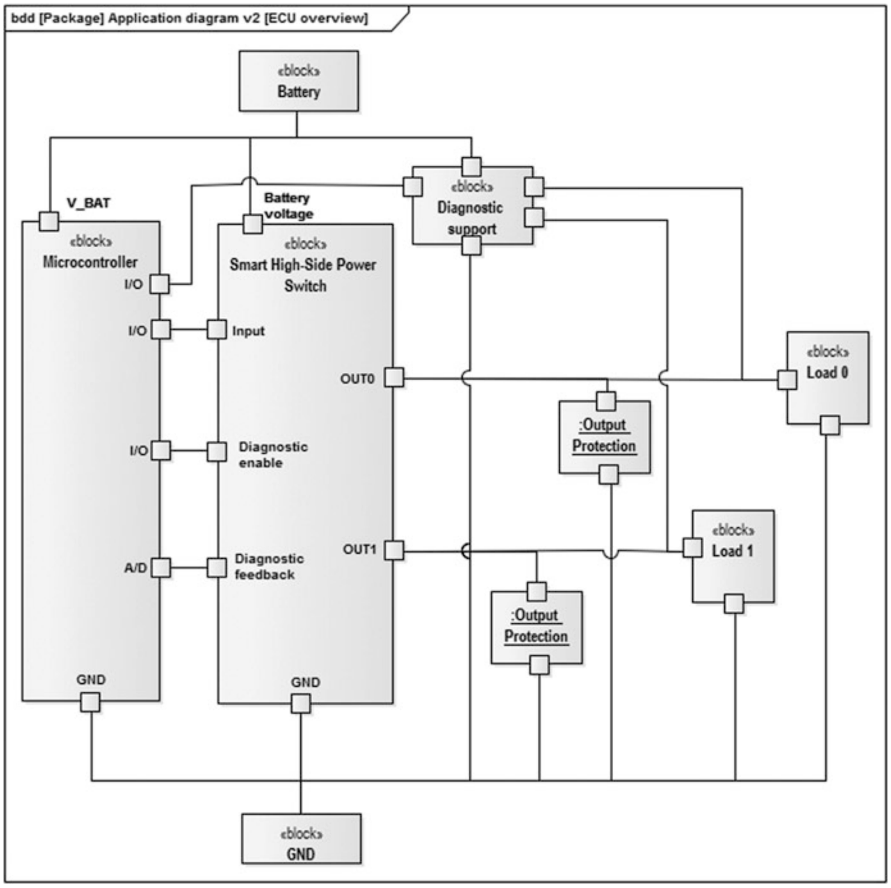

Fig. 4.3 ECU overview

that contain them and the activity or state machine diagrams that describe their characteristics.

The abstraction level used in the BDDs and IBDs is relative and strongly depends on the device being modelled as well as on the people that share this information. Our example, the BDD shown in Fig. 4.3, is addressed to people who do not have strong knowledge, or are not interested in electrical details. It demonstrates the interaction with the exterior devices and includes components that are very sensitive or can play a crucial role in safety-related scenarios. For example, the ground (GND) is specified because it is object to explicit requirements about losing the ground connection or having a short towards it. Instead, Fig. 4.4 provides details about the components of the devices that fulfil different functions specified in the requirements. If requirements raise the necessity to further detail a given component of the high-side switch (e.g. ESD protection block), another IBD will be developed. This IDB will contain the necessary details and it will be linked to the parent

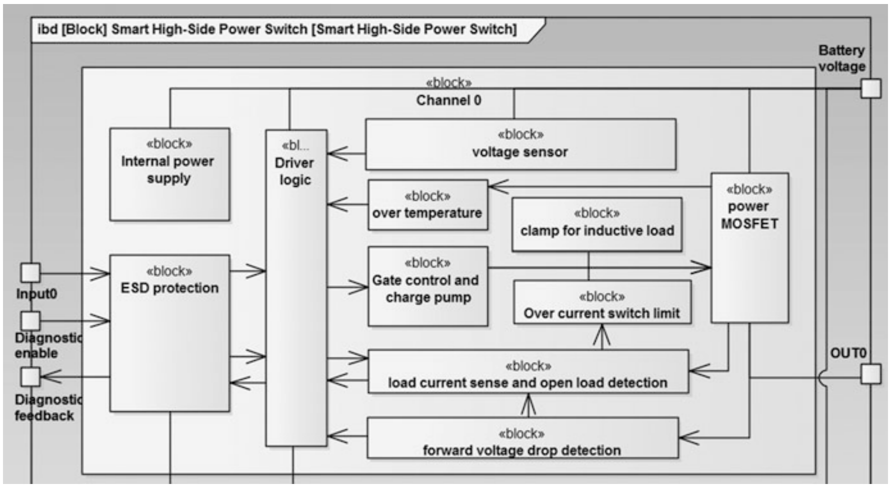
ibd [Block] Smart High-Side Power Switch [Smart High-Side Power Switch]
Battery voltage
«block» Channel 0
«block» Internal power supply
«bl... Driver logic
«block» voltage sensor
«block» over temperature
«block» clamp for inductive load
«block» power MOSFET
«block» ESD protection
«block» Gate control and charge pump
«block» Over current switch limit
Input0
Diagnostic enable
Diagnostic feedback
OUT0
«block» load current sense and open load detection
«block» forward voltage drop detection

Fig. 4.4 Internal block diagram of the high-side switch

diagram. Hence, a hierarchy of structure diagrams is constructed which allows a fast navigation through different levels of abstraction. Such flexible abstraction level representations have another great advantage: it gives the possibility to hide or show details depending on the confidentiality or the competence that readers have.

In the first stages of a project it is vital to have an overview about the application structure for the high-side switch. Further, the component can be refined using an IBD. The internal representation can be constructed inheriting detailed description of the relevant components, for example, the safety-related elements.

The interaction with the environment and the hierarchical connection, where the component can be represented as a black box as well as with necessary details, prove flexibility and represents a benefit using SysML modelling.

The model remains understandable at each level, by containing only the amount of detail relevant or desired. At the same time all requirements can be linked to the modelled structural elements. In case of a change at the requirement level, the impacted elements are easy to identify. If there are changes later in the design phase, it is fast to identify whether the requirements attached are still fulfilled and implemented.

Behavioural Modelling

Modelling safety protections and diagnostic functions can be achieved by using state machines and activity diagrams. AMS functionality modelling using SysML assumes a clear understanding of the behaviour from a high level of abstraction, which can be further implemented and verified. This means that any uncertainty about the behaviour of the device is solved in advance. If any uncertainty still exists during modelling this will be put in evidence by the modeller and solved by the

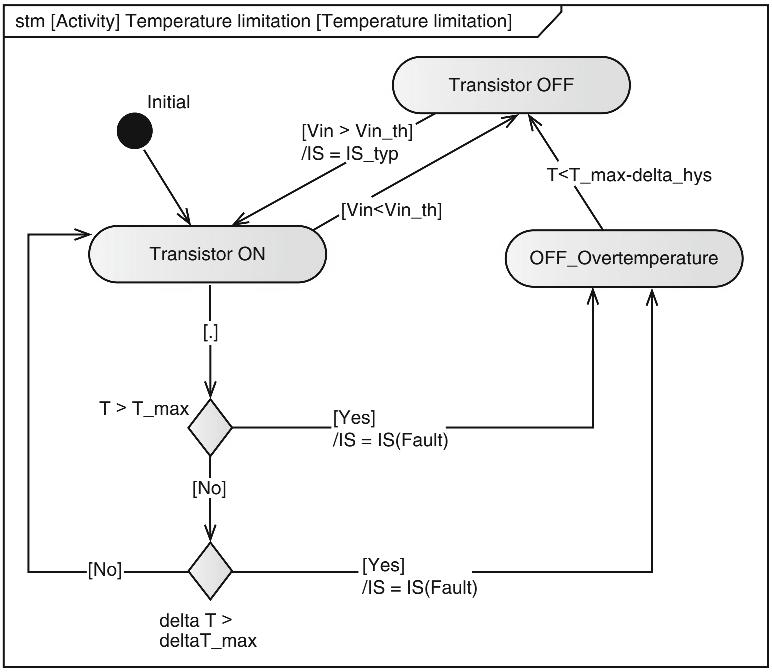

Fig. 4.5 Temperature limitation

interested parts. Therefore, while trying to make the behaviour easy to understand by modelling it, an implicit verification step is performed. With an easy-to-understand model misunderstandings and misinterpretations are easier and faster to find and to correct. There are two approaches used to describe these functionalities. The first one is to model each function separately while the second one models all the functions in the same diagram in order to catch all the aspects related to interactions that may occur among functions.

The description of a single function using state machines consists in modelling an AMS function with states and transitions. For each transition, it is specified the condition of the transition and the diagnostic information—a predefined value for the fault signal. The value of the diagnostic signal is maintained until the mechanism jumps to another state.

Figure 4.5 shows a SysML model of the over-temperature protection function. The behaviour of the transistor is modelled by three states: ON, OFF and OFF_Overtemperature. ON means that the switch is closed and the current flows through the load, while OFF means that the switch is open and the load current is zero. During the ON state the temperature is sensed and compared to a predefined threshold. In case that the temperature rises above a predefined threshold, the protection function reports this to the logic which commands the transition to the OFF_Overtemperature state for the transistor. The same behaviour occurs even

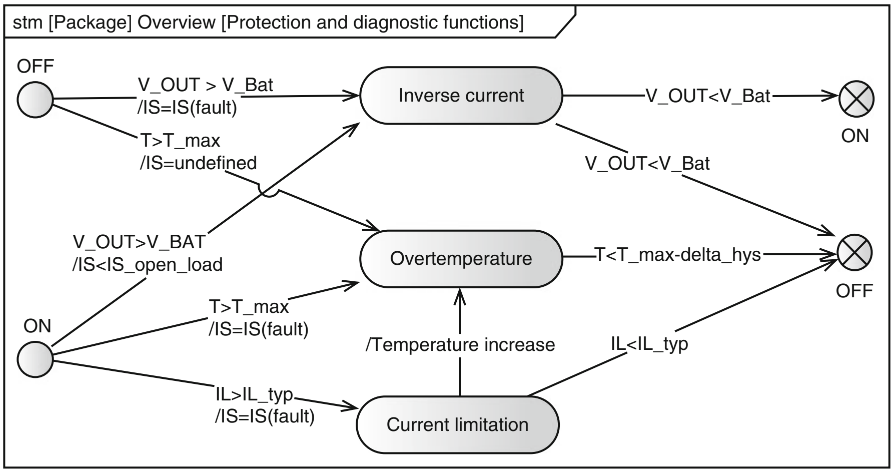

Fig. 4.6 Protection functions

when there is a fast increase in temperature, i.e. when the temperature change (deltaT) within a given interval of time is greater than a threshold (deltaT_max). At the same time the feedback current IS signalizes the faulty case to the uC. The thermal shutdown protects the high-side switch by turning the device completely off. In order to keep the IC from oscillating in and out of the thermal shutdown mode, the hysteresis logic is modelled. In this situation the chip goes back to OFF mode when the temperature value is below the min hysteresis value (delta_hys). A transition back to ON mode is done from OFF mode for the normal voltage conditions (when the input voltage is above the defined threshold to switch from OFF to ON).

The second approach for representing the behaviour is to include all the protection and diagnostic functions into one diagram in order to show how these functions interact and affect each other's behaviour. Figure 4.6 shows an example of three faulty situations and their effect—protection and diagnostic. A fault in the system can happen in both ON and OFF mode. A current limitation can lead to an increase of the temperature and this can lead to thermal shutdown.

Both behavioural models described and illustrated above represent only a small example of the entire behaviour. The complexity of the system, leads also to complex diagrams, where the level of details can be chosen depending on the purpose of the model as well on the target and intention of the person that makes the model.

Every element from the behavioural models (Figs. 4.5 and 4.6), as well as from the structural description (Fig. 4.4) represents a data structure with well-defined properties, like name, author, status, version, complexity, etc. One of the most important properties is represented by the related links, where the relationships between elements are specified. When an element is linked to its requirement, this relation will be available in the properties, containing all the information required:

name of the linked element, element stereotype (e.g. requirement) and connection type (e.g. realization).

A description of all the functionalities together shows how complex the behaviour of the whole device can become. It is relevant to show how the functions work independently as well as how they interact with each other and influence other functions. Compared to natural language description, problems such as an undefined state or transition become visible using state machine or activity diagram representations.

4.3.2 Airbag System

The second use case is also from the automotive domain and it highlights the methodology for the semi-formal modelling approach. An airbag system comprises sensors, actuators and an ECU. The ECU communicates with all sensors and actuators related to the system. Sensors not only include crash detection devices like pressure or acceleration sensors, but also buckle switches and seat occupancy detection; actuators include squibs for airbags as well as safety belt pretensioners.

The ECU receives information about possible crash situations from the sensors and based on internal decision, activates the actuators in case of a crash. The information received by the ECU is available from different types of sensors, like G-force sensors, pressure sensors as well as sensors indicating on which seats persons are present in the vehicle. This information can be either digital or analogue depending on sensor technologies and is transmitted to the ECU. When a crash is sensed, the control unit sends an electrical signal to the corresponding actuators, also known as squibs which will then deploy the airbags.

The main part of the system responsible for the interpretation of the sensors signals and decisions in case of a crash is the ECU. It comprises a microcontroller including the software running on it responsible for the decisional factor in case of a crash, various sensor interfaces, squib drivers, an independent safing engine evaluating sensor data, as well as many other components. For this chapter we assume that all sensor interfaces, squib drivers, and other support functions are integrated into a single component named Airbag IC.

The airbag IC receives the information from the sensors, which is translated to digital signals and communicated to the microcontroller via a Serial Peripherical Interface (SPI) interface. The microcontroller evaluates the received data, and triggers airbag deployment in case a crash is detected based on the received information. As a safety measure, the safing engine also evaluates the sensor data to check for a potential crash independently.

The airbag IC receives the deployment request from the microcontroller, confirmed by the safing engine and releases the squibs for airbags as well as for seatbelt pretensioners.

The combination and interaction of the hardware elements, both analogue and digital, inside of the airbag IC or the safing engine as well as its communication with the microcontroller on the ECU and with the sensors and actuators outside of the ECU as well as the safety relevance of the entire application considerably increases the complexity in all development phases of the product.

As for the protected high-side switch, the requirements specification for the airbag IC and the safing engine includes and highlights the safety requirements, mapped to diagnostic functions and bidirectional confirmation or rejection of crash detection between the microcontroller and the safing engine.

4.3.2.1 Modelling in SysML

Requirements Modelling

The requirement specification documents have been defined at the level of the airbag IC as well as at the level of the internal sub-systems. Due to the high complexity of the whole system we have selected a sub-system for analysis and modelling, namely the safing engine, motivated by its safety relevance.

The first step is the organization of the natural language requirements in models, depending on their types and relationships. Figure 4.7 depicts the top-level hierarchy of the requirements, classified in different categories: FSRs, normal operation requirements, implementation requirements, requirements coming from standards, chip package requirements and application information. The requirements are further split into sub-categories in order to ease for implementation, verification, validation and back-tracing from the results.

Organizing requirements into packages that correspond to various categories and stakeholders provides consistency with the specification document where the requirements are defined in natural language; it facilitates the configuration change management processes and offers support in organizing the further verification of requirements.

The FSRs include three general categories: the TSRs, the safety requirements related to architecture (for both software and hardware) and the safety-relevant use cases. The set of safety requirements presents all "requirements for the safety instrumented functions that have to be performed by the safety instrumented systems" (definition according to the standard IEC 61511). For example, we can consider an analog to digital converter (ADC) as a safety-critical architectural element. In order to fulfil this safety requirement, one possibility is to implement a second ADC as a redundancy mechanism, in order to mask any faulty behaviour of the converter. Another example for a safety architectural element is a supply voltage regulator for a digital logic circuit. A fail of the regulator may affect the computational logic. As a safety measure, under- and over-voltage mechanisms need to be implemented so that they will detect a fluctuation of the supply voltage and prevent the circuit from being damaged. The under- and over-voltage detection mechanisms should be supplied by different sources in order not to be affected by the same problem as the supply voltage regulator.

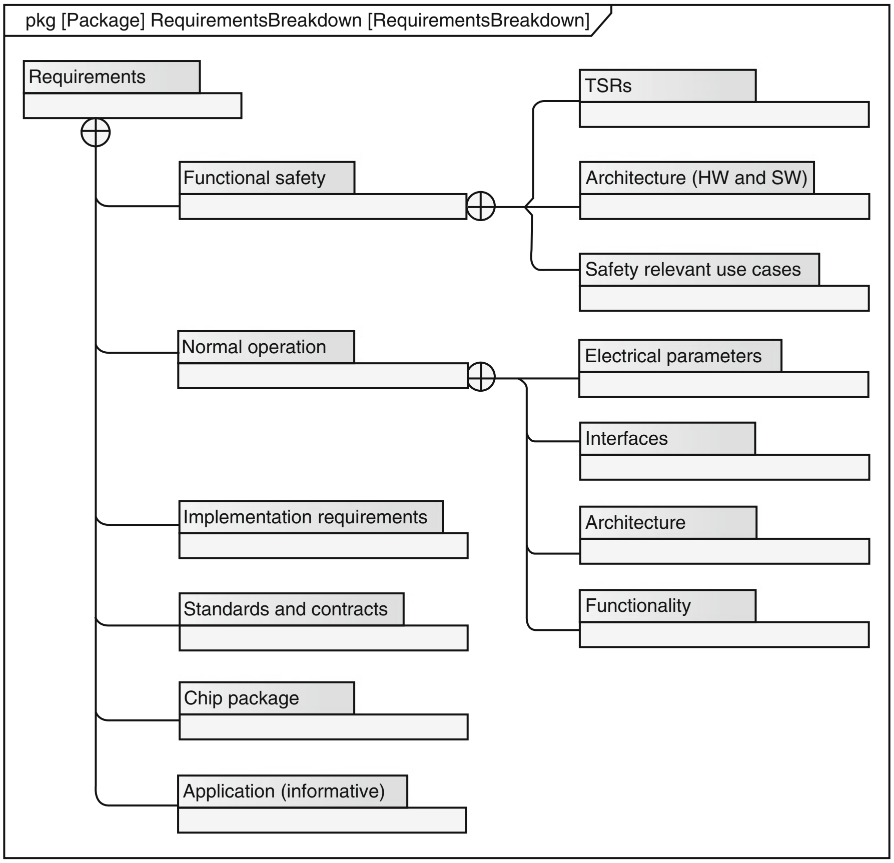

Fig. 4.7 Requirements breakdown for the Airbag Safing Engine

Another brake down of requirements is done for normal operation, based on application needs and specifics such as mandatory electrical parameters, information for the interfaces to the outside environment, architectural and functional requirements. The electrical parameters are usually listed in a table, with the minimum, typical and maximum values that the product needs to fulfil under certain defined operating conditions (e.g. for a given temperature range). A lot of electrical parameters are related to the safety and can be traced to the related safety requirements. As presented in the example above, the limit values for the under- and over-voltage will be specified in the table for parametric requirements.

Interface requirements shall include all the information necessary for the interaction with the outside world, i.e. what are the inputs and outputs, which types of inputs they are and what kind of information is communicated through these interfaces. For example, for the definition of a SPI communication, there are specific inputs and outputs well defined in the SPI standard and detailed depending on the application.

The architecture requirements represent a high-level view of the targeted structure—usually centred on the functionality—and will differ from the real implementation.

The functionality requirements describe all the needs for the behaviour of the system. They include, for example, operating modes (like unpowered, normal operation, safe mode, sleep mode, etc.), combinatorial and sequential behaviour (illustrated sometimes by waveforms drawings) or configuration options.

The requirements database can also include design and manufacturing requirements, compiled as implementation requirements, which are important for development and production of the device.

Information and requirements for the IC package must also be added, as well as requirements from standards that need to be fulfilled by the product. For example, EMC (Electro-Magnetic Compatibility) standards usually provide information for dedicated measurements during verification.

We suggest adding information about the use of the targeted product as application notes in a separate container. Unlike to other requirements, the notes are only informative but support a human reader in understanding the normative requirements and their motivation. Although tracing or verification of information categorized as Application (informative) is not needed, it's recommended to use a common requirements database to ease use and maintenance of all information. Application notes may help to do verification setups properly and—although not an explicit target of requirements modelling here—support the validation of requirements. Trace links between normative requirements and informative notes may be helpful.

The SysML package structure is used to create a requirements portioning model, which is useful for identifying a complete set of requirements, an essential part in the requirements definition. Each specification package contains the text-based requirements for that category. Relationships between requirements from different packages can be defined, for example by explicitly linking requirements from a safety standard (e.g. ISO26262) to the FSRs. The package is organized using a containment relationship, the representation allowing a compact way to view the hierarchical decomposition.

The requirements are compiled from different sources owned by different stakeholders. The structural decomposition of requirements categories is done using BDD, while the requirements for each category are defined using requirements diagrams.

Figure 4.8 depicts an example of requirements for functionalities that are suggested to be implemented as a state machine. The safing engine is required to work in four different modes: startup mode, diagnostic mode, normal mode and safe mode. The behaviour of each mode as well as the transitions between the modes are described in natural language. Startup represents a safe state where the configuration for the safing engine is done. In diagnostic mode checks of the entire functionality of the safing engine are performed to ensure proper functionality. The normal mode represents the normal working mode, where background checking is still performed.

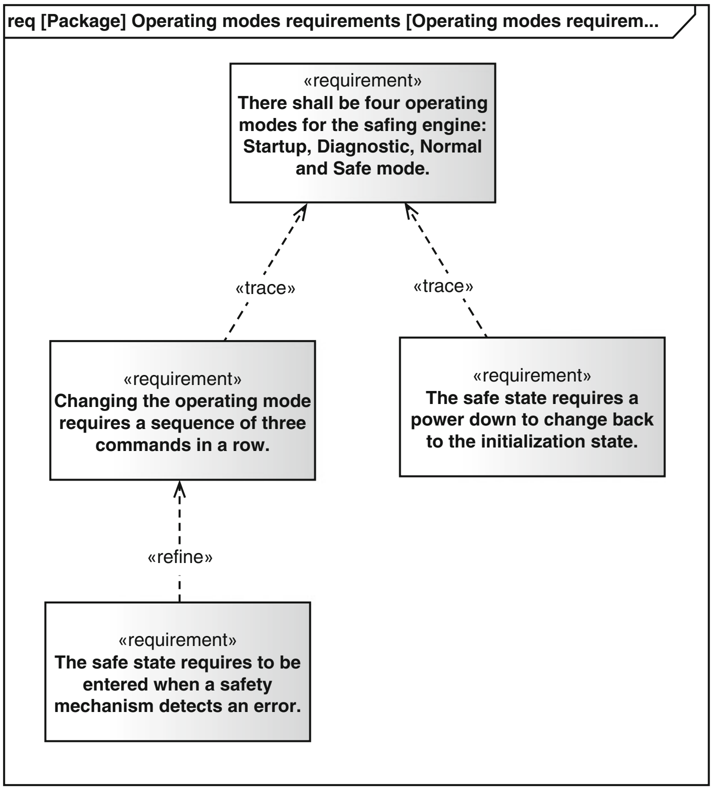
req [Package] Operating modes requirements [Operating modes requirem...
«requirement»
There shall be four operating modes for the safing engine: Startup, Diagnostic, Normal and Safe mode.
«trace»
«trace»
«requirement»
Changing the operating mode requires a sequence of three commands in a row.
«requirement»
The safe state requires a power down to change back to the initialization state.
«refine»
«requirement»
The safe state requires to be entered when a safety mechanism detects an error.

Fig. 4.8 Safing engine requirements example

If a fault is detected at any state, the safing engine will change to the safe mode, which can be exited only by a power down and power up back to the initialization.

There are three relationships types that are used to show how one requirement is related to another: nesting, derive and copy. And there are three types of relationships that can be used to connect a requirement to any type of model element, not only another requirement. These are satisfy, refine and trace relationships. In the context above we use the refine and trace relationships to show the relations between the requirements. These various types of relationships highlight explicitly the connections between different parts of a model as a way to maintain the consistency of the model. If a requirement is not traced to a model element for structure or behaviour, it should be checked weather the requirement is necessary or weather the model element is not yet available. Capturing the traceability within a SysML model enables the possibility to perform a downstream impact analysis. A behavioural or structural model can be easily identified when it depends on a requirement that has changed over time, an advantage of reducing the time and costs when implementing changes in a design over the system development cycle.

Structure Definition

Once the requirements are available in the modelling environment, the second step is the construction of the structural view of the system using BDDs and IBDs based on the natural language requirements. The main purpose is to reflect only the information described in natural language, without crossing the border towards design or implementation. For this there are three views taken into consideration: the context of the system—the application and interaction with the environment, the system with its inputs and outputs—analogue and digital pins, and the internal description—based on the desired level of detail described in the natural language requirements.

Figure 4.9 depicts the main ECU blocks embedded in the application context for the safing engine. The microcontroller as well as the safing engine receive information from the sensors via the sensor interface module, as well the information if the seat is occupied and if the seat occupant has the seat belt buckled or not. In case the microcontroller detects a crash based on the sensor data received and the safing engine confirms the crash, it will send a fire command to the squib drivers. When the safing engine detects a crash situation—also based on the sensor data received—and gets a confirmation from the microcontroller, it will enable the power supply for the squib drivers. Only when both entities detect a crash and confirm this situation to each other the airbag will be deployed during the time indicated by the requirements.

The model shows a high-level representation of the structure based on the requirements. When further refining the architecture, blocks as well as interfaces between blocks may change, e.g. to improve implementation efficiency by using blocks and interfaces as shared resources or to split blocks and interfaces as safety mechanisms.

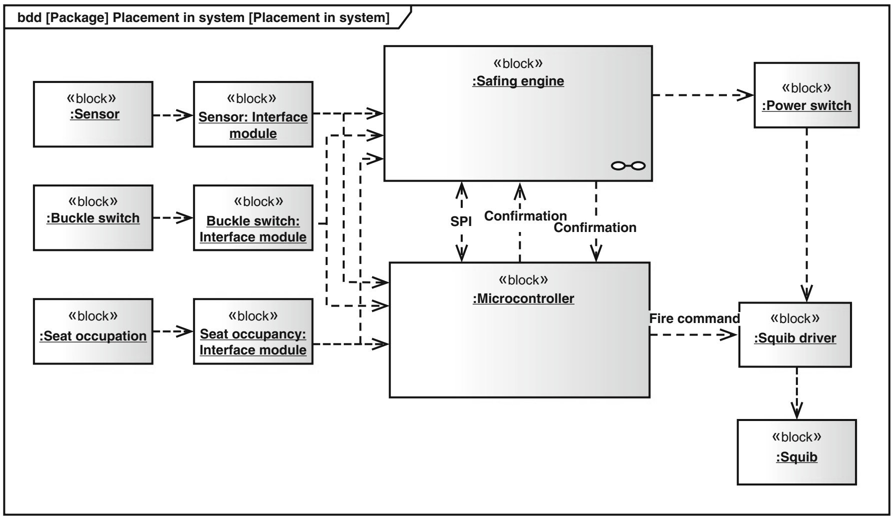

Fig. 4.9 Safing engine system interaction

The safing engine is further modelled based on the structural requirements using IBDs while the structure elements are linked to their requirements.

The main components of the safing engine are depicted as blocks. In order to depict the interconnections and communication flow between the blocks, dependency relationships are used. The diagram shows how sensor data passes through processing blocks to perform computations for crash detection. The blocks are further hierarchically decomposed as far as necessary to provide proper targets for explicit linking to requirements.

The majority of the blocks are defined in component libraries and then instantiated in the structural model view. The main aim of this representation is to provide a clear and simple view of the safing engine interactions with outside elements and the type of interactions and further to describe the decomposition and the relations between the internal components.

Behavioural Description

The requirements listed in section A represent a short example of the functional requirements of the operating modes of the safing engine. The functionality expressed by the requirements is modelled using a state diagram. It comprises four modes: initialization mode, diagnostic mode, normal mode and safe mode. The four modes can be modelled using state blocks in a state diagram (as in the example illustrated in Fig. 4.10) and further used as a state machine element in the next higher level of the model hierarchy.

The power up of the device has been represented by as an entry point item. When the device is powered up, it enters automatically in the initialization state. A transition from initialization mode to diagnostic mode will be performed only by receiving three specific commands in a row, which are described separately. The same applies for the change from the diagnostic mode to normal mode. The safe mode will be reached as a result of a fault detection in any of the other modes or when a specific unlock sequence for entering the safe mode is received from the microcontroller. The natural language requirement states that the safe mode can be exited only by a power down of the system. In fact any state can be exit by a power down, but the specific description of these requirements for the safe mode is intended as a verification requirement explicitly described.

The main issue detected when modelling the requirements using the state diagram, was the incompleteness of natural language description of the transitions from one state to another by a specific sequence of three commands—the requirement "Changing the operating mode requires a sequence of three commands in a row". The order of the commands as well as specifying which commands are needed is described separately in the requirements for the communications.

It was not specified in natural language requirements how to react in case of wrong commands or wrong command sequence. Without a specification for these cases it is unclear whether the system shall keep the current state or enter safe mode.

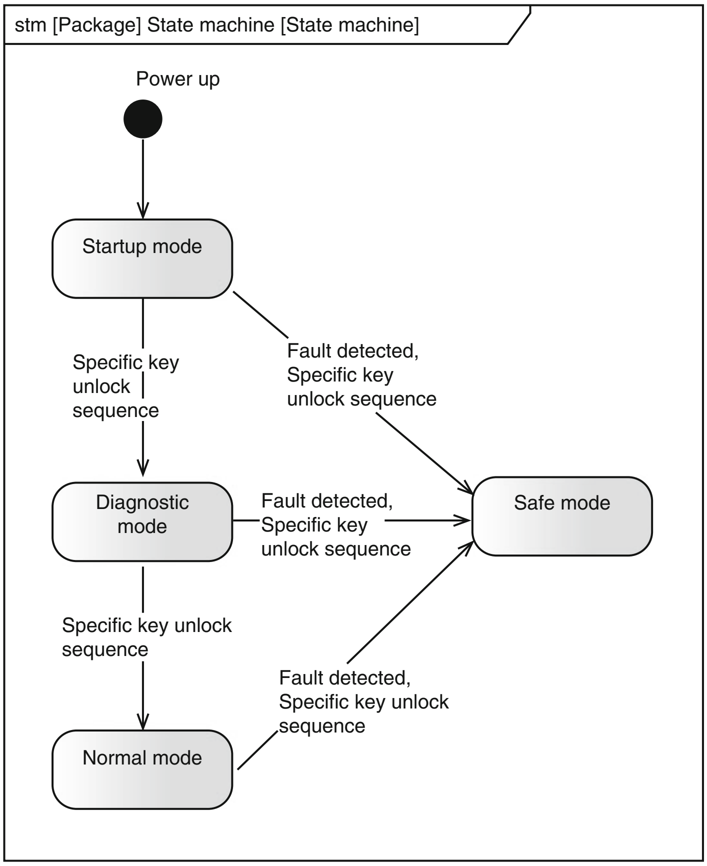
stm [Package] State machine [State machine]
Power up
Startup mode
Specific key unlock sequence
Fault detected, Specific key unlock sequence
Diagnostic mode
Fault detected, Specific key unlock sequence
Safe mode
Specific key unlock sequence
Fault detected, Specific key unlock sequence
Normal mode

Fig. 4.10 Safing engine state machine functionality

Natural language requirement descriptions are commonly complemented by drawings reflecting structure or behaviour. Typical examples for AMS designs are block diagrams, state diagrams, truth tables, waveform diagrams and even transistor level circuit illustrations. For most of these drawings we recommend to use SysML to benefit from the standardized syntax and semantics. Only those illustrations that have no direct representation in SysML and cannot be translated to SysML without losing significance and clarity should be left in their original form. But they still should be captured by the modelling environment to allow consistent linking between requirements and other SysML objects.

4.4 Conclusions

The diagram-based representation of the requirements allows a better organization, processing and classification of the requirements compared to classical representations like natural language text. By exploiting the SysML capabilities the requirements can be linked to each other according to the relation that exists between two requirements. Moreover the requirements can be linked to structure elements of the device and behaviour models. Such links facilitate the impact analysis of the requirements and guide changes that can occur in different phases of the development.

The diagrams used for modelling the structure of the overall system and the internal structure of components permit a flexible level of abstraction. This flexibility allows hiding or revealing details depending on complexity of the device, the knowledge and expertise of the involved people and the level of information confidentiality.

These features were illustrated by modelling two AMS hardware components: a protected power switch and a safety-relevant module of an airbag system—a safing engine related to the decision to fire squibs for the airbag deployment.

For both SysML use-case examples—the protected power switch and the safing engine—the behaviours of the protection functions and the entire activity, respectively, were modelled by using state machines. State machines offer the possibility to reflect the specifications of events that trigger a given functionality and the actions that are taken once the trigger conditions are fulfilled. All types of diagrams (behavioural, structural and requirements) were linked to each other in order to ensure the integrity of the requirements set, as illustrated in the sections before.

By creating a semi-formal model ambiguities are recognized easier and avoided compared to natural language. The explicit allocation of requirements to blocks or sub-systems encourage or even require to perform a basic feasibility check and hence improve the quality of the specification.

Another possible usage of this model is to connect it to concept and design models which can be simulated and verified. The link between representations as well as requirements needs to be maintained. Furthermore, by using SysML it is possible to specify explicitly test cases that are not deduced from other requirements and link them to different parts in the model that will be refined further and used for verification, especially to ensure that safety requirements are fulfilled.

Acknowledgments The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement Nr. 295311 and the Austrian Research Promotion Agency FFG under the program "Forschung, Innovation und Technologie für Informationstechnologien (FIT-IT)".

References

1. OMG System Modeling Languages. http://www.omgsysml.org/. 13 Feb 2014 (01 May 2015)
2. International Council of Systems Engineering. http://www.incose.org/. 02 Apr 2014 (01 May 2015)
3. Collins, M.: Dependable Embedded Systems, Topic: "Formal Methods." Carnegie Mellon University (1998). https://www.ece.cmu.edu/~koopman/des_s99/formal_methods/
4. Broadfoot, G.H.: ASD case notes: Costs and benefits of applying formal methods to industrial control software. In: FM 2005: Formal Methods. LNCS, vol. 3582, pp. 548–551. Springer (2005)
5. Eltahir, S., Musa, M.: On practicality of using integrated semi-formal modeling tools. In: The International Arab Conference on Information Technology, 2008
6. Unified Modeling Language (UML). http://www.uml.org/. 14 Feb 2014 (01 May 2015)
7. Feiler, P.H., Gluch, D.O.: Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley Professional (2012)
8. Architecture Analysis & Design Language (AADL). http://www.aadl.info/aadl/currentsite/ (2012) (01 May 2015)
9. EAST-ADL. http://www.east-adl.info/ (2014) (05 May 2014)
10. Modeling and Analysis of Real-time and Embedded systems (MARTE). http://www.omg.org/omgmarte/Tutorial.htm (2008) (01 May 2015)
11. Evensen, K.D., Weiss, K.A.: A comparison and evaluation of real-time software systems modeling languages. Presented at the Aerospace Conference, Georgia, Atlanta, 2010
12. Helming, J., Koegel, M., Schneider, M., Haeger, M., Kaminski, C., Bruegge, B., Berenbach, B.: Towards a unified requirements modeling language. In: Fifth International Workshop on Requirements Engineering Visualization (REV), 2010, pp. 53–57
13. Kaiser, B., Klaas, V., Schulz, S., Herbst, C., Lascych, P.: Integrating system modelling with safety activities. In: SAFECOMP Proceedings, 2010
14. Adedjouma M., Dubois, H., Maaziz, K., Terrier, F.: A model-driven requirement engineering process compliant with automotive domain standards. In: Model Driven Tool and Process Integration, 2010
15. Sontos, M.d., Vrancken, J.: Model-driven user requirements specification using SysML. J. Software **3**, 57–68 (2008)
16. Wang, B., Baras, J.S.: Model-based design framework for wireless sensor networks using SysML, Simulink and Modelica. https://www.src.org/library/publication/p061828/. 28 Oct 2011
17. Mueller, W., He Da, Mischkalla, F., Wegele, A., Whiston, P., Peñil, P., Villar, E., Mitas, N., Kritharidis, D., Azcarate, F., Carballeda, M.: The SATURN approach to SysML-Based HW/SW Codesign. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2010, pp. 506–511
18. Mischkalla, F., He Da, Mueller, W.: Code generation for QEMU/SystemC Cosimulation from Cosimulation from SysML. Presented at the MeCoES Workshop, 2012
19. Waseem, R.: Accelerating High-Level SysML and SystemC SoC Designs. http://www.design-reuse.com/articles/17562/high-level-sysml-systemc-soc-designs.html (01.06.2015)
20. Infineon Technologies AG: Product_Brief: BTT6050-2EKA (Truck device) PROFET™ + 24V. http://www.infineon.com/dgdl?folderId=db3a30431400ef68011421b54e2e0564&fileId=db3a30433784a0400137984c3a63271b&intc=0120035 (01.06.2015)
21. Infineon Technologies AG: Introduction to PROFET™. http://www.infineon.com/dgdl/Introduction+to+PROFET%E2%84%A2.pdf?folderId=db3a30431400ef68011421b54e2e0564&fileId=db3a304332ae7b090132b527d9173083 (01.06.2015)
22. Infineon Technologies AG: Protected high side drivers. In: Bridging Theory into Practice – Fundamentals of Power Semiconductors for Automotive Applications, 2nd edn. pp. 125–149. Infineon Technologies AG, Munich (2008)

23. Burton, D., Delaney, A., Newstead, S., Logan, D., Fildes, B.: Effectiveness of ABS and vehicle stability control systems. RACV Research Report (April, 2004). http://www.monash.edu.au/miri/research/reports/other/racv-abs-braking-system-effectiveness.pdf (24 Apr 2014)
24. Petin, J.-F., Evrot, D., Morel, G., Lamy, P.: Combining SysML and formal models for safety requirements verification. In: International Conference on software & Systems Engineering and their Application, 2010
25. Bryans, J., Payne, R., Holt, J., Perry, S.: Semi-formal and formal interface specification for system of systems architecture. In: IEEE International Systems Conference (SysCon), 2013, pp. 612–619
26. Jarraya, Y., Soeanu, A., Debbabi, M., Hassaine, F.: 10-Automatic verification and performance analysis of time-constrained SysML activity diagrams. In: IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2007, pp. 515–522
27. Gnaho, C., Semmak, F., Belkaid, Y., Laleau, R.: Goal-based requirements engineering in topcased environment. Presented at the TopCased Days, Paris, France, 2011. http://gforge.enseeiht.fr/docman/view.php/52/4276/A1-LACL.pdf (05.05.2014)
28. Hove, D., Goknil, A., Kurtev, I., Berg, K., Goede, K.: Change impact analysis for SysML requirements models based on semantics of trace relations. Presented at the ECMDA Traceability Workshop, Enschede, Netherlands, 2009
29. Favaro, J., Koning, H.-P., Schreiner, R., Olive, X.: Next generation requirements engineering. http://www.intecs.it/PDF/NextGenRE_INCOSE_FINAL_2012.pdf (01.06.2015)
30. Bachhuber, A.: Requirements engineering in the product life cycle of continental automotive. Presented at the RFConf, Munich, German, 2013. http://www.hood-group.com/fileadmin/project/reconf/VortraegePDF/mm3_bachhuber_requirements_engineering_in_the_product_life_cycle_of_continental_automotive.pdf (01.06.2015)
31. Infineon Technologies AG: Smart high side switch. http://www.infineon.com/profet (01.06.2015)