Joint Specification and Testing of Safety and Security Requirements

Sadegh Sadeghipour[1], Holger Schlingloff[2], Mirko Conrad[3], Harald Schülzke[3]

[1] ITPower Solutions GmbH, Berlin, Germany

[2] ZeSys e.V. and Fraunhofer FOKUS, Berlin, Germany

[3] samoconsult GmbH, Berlin, Germany

Abstract

Violation of safety or security in modern highly networked and automated devices and functions, such as those used for the Internet of Things, Industry 4.0, and autonomous driving, can lead to catastrophic consequences for people and the environment. Therefore, the development process of embedded systems and software is associated with demanding requirements regarding safety and security.

While the discipline of safety engineering is well established and supported by international standards like IEC 61508, ISO 26262, and EN 5012x, security engineering and its interaction with the safety process in the field of embedded systems is still in early phases of its development.

In this paper we present a methodology for the joint specification of safety and security requirements of embedded systems, and the derivation of test cases. Currently, safety and security are treated in two separate engineering processes. The advantage of a process for specification both safety and security at the same time is that possible redundancies and inconsistencies between safety and security requirements can be identified at an early stage.

The core of the methodology presented here is a domain-specific language (DSL) called LESS (Language for Embedded Safety and Security), which is based on natural language templates often used in requirements engineering. With the aid of a few simple rules and a small set of keywords users can define safety and security requirements in a formalized way without any need for a difficult-to-understand mathematical or a complex graphical notation.

We also present a set of methods that form the basis for implementing semi-automatic procedures for analyzing and refining requirements as well as deriving test cases from them. These methods are based on an analysis of the syntactical structure of the safety and security requirements expressed in LESS, and on the design of controlled conversations with the user. The results of the conversations are used to achieve further semantic information needed for the analysis, refinement and derivation activities mentioned above. Some of the developed methods have been implemented as prototypes and applied to case studies from automotive and medical technology.

Due to the easy-to-learn and well-understandable domain-specific language LESS, as well as the controlled wizard-like conversations with the user, the methodology presented here possesses a high potential to be used in all industrial sectors where safety- and security-related applications are developed.

Keywords: Safety and Security Co-Engineering, Language for Embedded Safety and Security (LESS)

---

* The described activities were conducted as part of the EmbeddedSafeSec project (Mar. 2021 – Dec. 2022). The EmbeddedSafeSec project was funded by the Investitionsbank Berlin (IBB) program for the promotion of research, innovation and technology – ProFIT – and by the European Regional Development Fund (ERDF).

Contents

1 Introduction

Current industrial trends such as Internet of Things, Industry 4.0, and autonomous driving, introduce new levels of automation and networking. In this context the violation of safety (i.e., the protection of the environment from the danger posed by a system) or security (i.e., the protection of the system from attacks by the environment) may lead to catastrophic consequences. Since embedded systems are the heart and brain of the systems mentioned above, safety and security engineering of embedded systems is a topic with outstanding significance for the respective development process.

The discipline of safety engineering is well researched and methodologically established in most industrial sectors that develop and use embedded systems, supported by internationally recognized standards such as IEC 61508 (Functional safety of electrical/electronic/programmable electronic safety-related systems), ISO 26262 (Road vehicles – Functional safety) [1], and EN 50128 (Railway applications - Communication, signaling and processing systems – Software for railway control and protection systems). The same can be said for information security management of general IT systems and applications, supported by corresponding international standards such as the ISO 27000 series on information security. Domain specific cybersecurity standards are also emerging, such as ISO/SAE 21434 (Road vehicles - Cybersecurity engineering) [2].

However, in the context of highly networked embedded devices and applications, the interplay between security and safety and their joint engineering and assurance are still questions lacking industrially applicable and satisfying solutions.

The electronic systems built into the devices must be able to communicate with different software versions of various network nodes, possibly unknown at the time of development. This partly requires the adaptation and reloading of the software during runtime. Furthermore, a typical embedded system is exposed to multiple variations of the physical and digital environment (pressure, temperature, friction, communication buses, digital inputs and outputs, etc.). All these characteristics hide potential hazards for safety and threats for security. The interaction between safety and security also adds to the complexity of the situation: Indeed, a breach of data security can

lead to a hazard for functional safety, e.g., intrusion of an unauthorized person into a Car2X network and manipulation of the communication between the vehicles in the network. A striking example of such an intrusion was published in [3], which brought the topic of the interplay between safety and security to public attention.

Furthermore, compliance with safety and security can lead to conflicting requirements for the system to be developed. For example, the encryption of system input data may be a necessary requirement from the security point of view. However, due to the execution time required for decrypting data, this may conflict with a possible safety requirement, according to which the input data must be processed in very short cycles. On the other hand, the fulfillment of safety and security in certain areas can place similar or same requirements on the system to be developed, e.g., a plausibility check of the input values of a component. These examples show that a separate engineering of safety and security can lead to late discovery of conflicts as well as duplication of activities, which in turn can lead to unnecessary effort and costs, and also to possibly high development risks.

In this paper we address a part of the challenge mentioned above. We present a methodology for the joint specification of safety and security requirements of embedded systems, and subsequent quality assurance methods. At the core of our methodology is a domain-specific language (DSL) called LESS (Language for Embedded Safety and Security), which allows the specification, refinement and analysis of safety and security requirements as well as the generation of test cases from them.

The work presented in this paper has been accomplished as part of the EmbeddedSafeSec project [4] concerned with various aspects of safety & security co-engineering for embedded systems. In addition to the methodology presented here, a process model for safety and security co-engineering of embedded systems was developed within the project and presented at SAEC Days in June 2022 [5].

2 Related Work

A large part of the research concerning the joint safety and security engineering deals with joint hazard and threat analysis methods. These extend either 1) an existing hazard analysis method of the safety process to a threat analysis method for the security process, or 2) combine the existing methods of both disciplines, or 3) propose an alternative method that is substantially different from the existing methods. To the first group belong for example the Security Aware Hazard Analysis and Risk Assessment [6], abbreviated to SAHARA, and the Failure Mode, Vulnerabilities and Effect Analysis Method [7], abbreviated to FMVEA. They extend the well-known safety analysis methods HARA and FMEA by including system threat information according to Microsoft's STRIDE model [8], [9] (STRIDE = Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege). The second group includes methods like Failure-Attack-CounTermeasure (FACT) [10] and Extended Fault Tree (EFT) [11]. They are based on a combination of fault trees from safety engineering and attack trees from security engineering. The methods of the third group propose a combined analysis of safety and security in the concept phase of system development using model-based and/or system-theoretical approaches [12], [13], [14]. While conventional hazard and threat analysis focuses on a component or a system, the starting point of the analysis here is the interaction between the system components and involved systems including their environment. In [15], Pereira et al. present a method for specifying safety and security requirements by constraints and identifying conflicts as well as similarities between them. While the analysis provided by this method focuses on conflict identification, our approach also delivers solutions for refining requirements and generating test cases from them.

A main result of the hazard and threat analyses are safety and security goals as well as an initial set of functional safety and security requirements to be fulfilled by the system under development. Therefore, a process of joint safety and security engineering may start with one of the analysis methods mentioned above and continue to specify the safety and security goals and requirements using the methodology presented in this paper.

The objective of our methodology is a systematic and tool-supported specification, refinement and analysis of requirements as well as test generation. This objective is also shared by the research fields *formal specifications* or *formal methods* and *controlled natural languages*. While they aim at a general description method for any kind of requirements, the focus of the methodology presented in this paper is restricted to safety and security requirements.

The field of formal methods (see e.g. [16]), existing almost from the beginning of computer science, deals with the description of system and software requirements often based on first order predicate logic (e.g., B-Method [17]), higher order predicate logic (e.g., Z [18]), or temporal logics (e.g., [19]). However, due to their complexity and their distance to the experience of software engineers they could not be established in industrial practice.

In contrast to formal methods the field of controlled natural languages does not aim at a mathematical formalization of requirements, but at a kind of standardized expression as near as possible to the natural language and the wording of requirements used by system and software engineers [20]. As a consequence, templates introduced by different controlled natural languages have vastly been adopted for the specification of requirements in different industrial sectors.

We share the objective of being near to the industrial practice with the field of controlled natural languages. Therefore, we designed our domain-specific language LESS (Language for Embedded Safety and Security requirements) as a controlled natural language for the specification of safety and security requirements. At the same time LESS uses some logical operators from predicate logic in order to precisely express the logical conditions used in many safety and security requirements. An approach for functional requirements which is similar to LESS, called MARS, is presented in [21].

3 The Domain-Specific Language LESS

The objective of the domain-specific language LESS is to define safety and security requirements with the aid of a few simple rules and a small set of keywords in a formalized way, without any need for a difficult-to-understand mathematical or a complex graphical notation.

At a high level LESS is based on a template which resembles the general grammatical structure of a sentence in natural language. Only sentences following this template are valid LESS sentences or terms. This is also the basic concept of controlled natural languages. One of the first works concerning requirements in this field is the one created by Chris Rupp [22]. The template presented in her book is a simple and general, and at the same time very useful, structure which can be used for any kind of software requirement in any industrial field (see Figure 1).

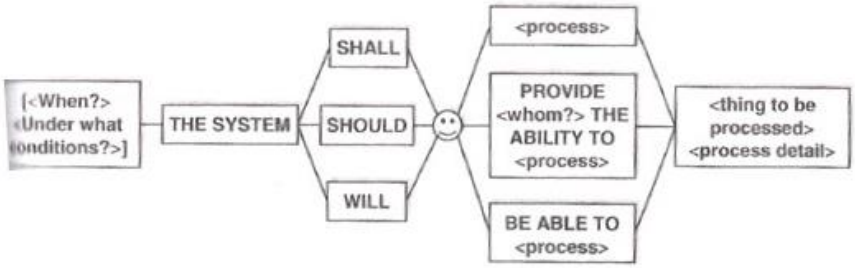

Figure 1 Rupp-Template [21]

Most templates developed and presented later are based on the Rupp template. They often represent a more detailed structure and allow to express different kinds of functional and technical requirements. For designing the LESS template, we considered the Rupp template as well as some more detailed ones developed by different authors, especially the template presented in [23].

3.1 LESS Template

Since the developed domain-specific language serves to express a specific kind of requirements, namely the safety and security ones, the LESS template does not possess a high expressiveness for other types of requirements. For example, due to the strictness of safety and security requirements, LESS allows only *SHALL* as modal verb, neglecting *SHOULD* and *WILL*, as proposed by Rupp. Therefore, a class of requirements, often denoted as "nice to have" features, cannot be expressed in LESS. At the same time, in order to achieve the precision required in the field of safety and security engineering, a formal syntax for logical and arithmetical terms is defined and embedded into the LESS language. As a consequence, the whole language was descripted in EBNF (extended Backus-Naur form).

Figure 2 shows the LESS template with its 13 positions. The bold font limited in angular brackets at the upper part of each position rectangle shows the non-terminal elements of the syntax. The words written using Capital letters in each rectangle denote the keywords which can be used in the corresponding position. All positions except for position 5 and 6 are optional.

3.2 E-Gas Case Study

A LESS specification document containing the safety and security requirements of a project consists of two parts. In the first part the vocabulary used in the requirements is defined, e. g., notions for process verbs, components, states, variables, object attributes, etc. The second part of a specification document contains the individual requirements, each of them described by a LESS term. It may be augmented by additional information such as the requirement identifier, the identifier of the parent requirement, the requirement classification, and its integrity/criticality.

The 'Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units' (E-Gas concept) [24] has been used as one case study to evaluate the applicability of LESS and the LESS methodology. The E-Gas concept contains a variety of functional and safety requirements for engine control units at different levels of abstraction. As part of the evaluation, a set of safety-related requirements from the E-Gas concept have been converted into terms compliant with the LESS Template (see **Fehler! Verweisquelle konnte nicht gefunden werden.**).

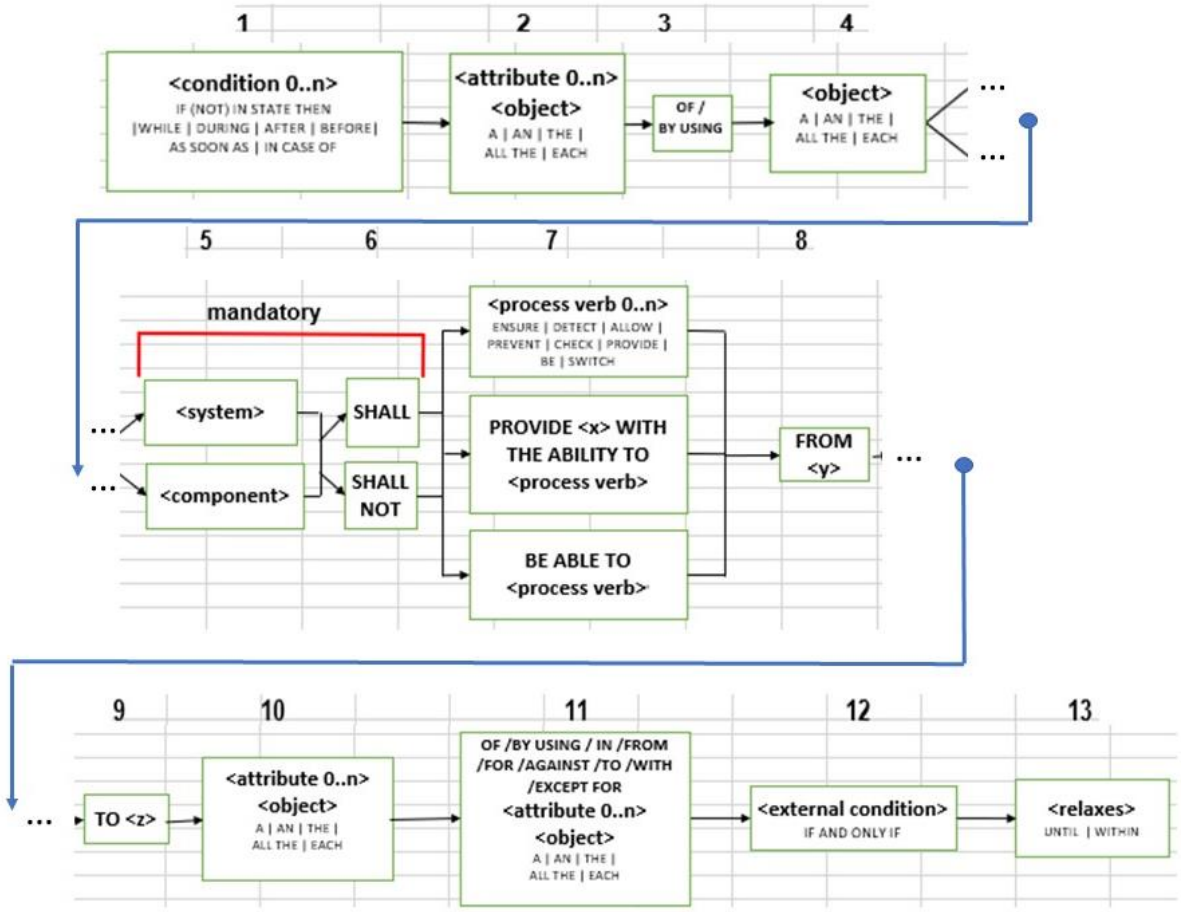

Figure 2 The LESS template

Example 1 (E-Gas)

EGAS_e-105
[SReq-01*] *'Sensors shall be plausibility checked' (Component: Drive pedal)*

In a first step the requirement was manually converted into a LESS compliant term in a way that preserves its meaning as much as possible. As the LESS template calls for mentioning the subject explicitly in position 5, [SReq-01*] was reworded into *'The Drive_Pedal SHALL check the sensor_ signals of the Drive_Pedal for plausibility'*. The mapping to the elements of the LESS DSL is shown below:

ID	…	5	6	7	…	10	11	…
[SReq-01]		The Drive_Pedal	SHALL	check		the sensor_signals of the Drive_Pedal	for plausibility.	

Example 2 (E-Gas)

EGAS_e-110
[SReq-06*] *'A safety concept shall be implemented in the engine control unit which detects and confirms undesired states of a high driving torque or an unintended acceleration. In case of a fault the engine control unit shall switch to a safe state.' (Component: Engine control unit)*

As the above requirement is not atomic (singular), it does not possess the corresponding characteristic required for safety requirements in ISO 26262-8:2018, clause 6.4.2.4.c [1]. As a consequence, it was first divided into 5 atomic sub-requirements which were converted into LESS compliant terms as follows:

ID	1	…	5	6	7	…	10	11	…
[SReq-06a.1]			The Engine_Control_Unit	SHALL	detect		undesired states	of High_Driving_Torque.	
[SReq-06a.2]			The Engine_Control_Unit	SHALL	confirm		undesired states	of High_Driving_Torque.	
[SReq-06a.3]			The Engine_Control_Unit	SHALL	detect		an Unintended_ Aacceleration.		
[SReq-06a.4]			The Engine_Control_Unit	SHALL	confirm		an Unintended_ Aacceleration.		
[SReq-06b]	In case of a fault		The Engine_Control_Unit	SHALL	switch to		a safe state.		

Of course, in the case no guideline for writing requirements is used, not all arbitrary safety or security requirements can be converted into a LESS compliant terms with a reasonable effort. In this case the grammar of the used language should allow the user to integrate such requirements into the same specification document in order to avoid the problems associated with multi-sourcing of the requirement specification. For that the LESS grammar offers the quotation mark symbol. As example we consider the following E-Gas requirement.

Example 3 (E-Gas)

EGAS_e-107
[SReq-04*] *'Torques affecting requirements of other ECUs shall be protected in a signal compound of the engine control unit.' (Component: Engine control unit).*

We consider the slightly reworded form of this requirement, where the subject is mentioned explicitly:
'The engine control shall protect torques affecting requirements of other ECUs in a signal compound.'

The phrase *'torques affecting requirements of other ECUs in a signal compound'* cannot be expressed in the LESS template. Therefore, as shown below, this phrase is set into quotation marks and is not interpreted by the analysis tool.

ID	…	5	6	7	String
[SReq-04]		The Engine_Control_Unit	SHALL	protect	"torques affecting requirements of others ECUs in a signal compound"

Example 4 (VAD)

We also show in the following an example from the second case study, which concerns the development of a ventricular assist device (VAD) as a medical technology application. The example describes a security requirement and contains some positions of the LESS template which were not used in the former examples.

Req_BH40 *'The clinical user interface shall be able to switch from manual to auto mode, if and only if the user is logged in.'*

In order to be fully compatible to LESS template the requirement was slightly rephrased and formulated as a LESS term as follows:

ID	…	5	6	7	8	9	…	12
[Req_BH40]		THE clinical_UI	SHALL	BE ABLE TO switch	FROM manual_mode	TO auto_mode		IF AND ONLY IF THE user IN logged_in

4 Specification and Testing Methods Based on LESS

The methods for semi-automatic procedures to analyze and refine safety and security requirements as well as deriving test cases from them are based on the following steps:

1. Analysis of the syntactical structure of the safety and security requirements expressed in a LESS specification document.
2. Designing controlled conversations with the user in order to achieve further semantic information needed for the analysis, refinement and derivation activities mentioned above.
3. Processing of the user's response and generating the result, which can be a consistency or completeness verdict, a refined requirement, or a test case.

In the following we describe for each of the following subjects one exemplary method:

- Detailing and refinement of safety and security requirements
- Consistency and completeness analysis of safety and security requirements
- Deriving test cases from safety and security requirements

4.1 Detailing and Refining Safety and Security Requirements

According to the terminology used in the fields of requirements engineering and safety engineering we consider the refinement relationship between different levels of safety or security requirements as shown in Figure 3.

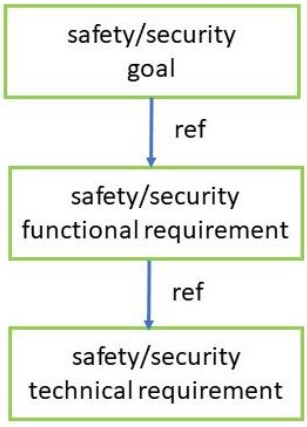

Figure 3 Refinement relationship between different levels of safety/security requirements

Our approach on refinement is based on an interactive process, where the machine supports the human by asking relevant questions. This way, we can use ample techniques for analyzing and transforming requirements without the need of formal tool qualification; the responsibility at any moment rests with the safety/security engineers.

Example 5 (E-Gas)

One of the rules for the interaction wizard is the following: “In case of a vaguely formulated condition, ask for a more detailed condition” Here is a sample dialogue using this rule:

```
[SReq-06b] ‘In case of a fault the Engine_Control_Unit SHALL switch
to a safe state.’
(Functional Safety Requirement, ASIL B)

[SReq-06b] refers to the case of a ‘fault’.
Would you like to further define a ‘fault’? [Yes/No/Later]

> Y
> ‘fault’ := ‘Unintended_Acceleration’

[SReq-06b.1] ‘In case of Unintended_Acceleration the
Engine_Control_Unit SHALL switch to a safe state.’
(Functional Safety Requirement, ASIL B)
```

As a result, the refined requirement [SReq-06b.1] was added to the requirements section of the LESS specification document. It was linked to its parent requirement [SReq-06b] and inherits the integrity level (ASIL) from its parent requirement.

Example 6 (E-Gas)

As another example, here is a dialogue exemplifying the rule “In case a requirement mentions a safe state, ask for detailed definition of the safe state”:

```
[SReq-06b.1] ‘In case of Unintended_Acceleration the
Engine_Control_Unit SHALL switch to a safe state.’
(Functional Safety Requirement, ASIL B)

[SReq-06b.1] refers to a safe state. Would you like to detail the
safe state mentioned in [SReq-06b.1]? [Yes/No/Later]

> Y
> [SS-01] := Injection driver of the gasoline engine is switched off
  (Safe State)

Would you like to review / update the original requirement [SReq-
06b.1] as well? [Yes/No/Later]

> Y
> [SReq-06b.1.1] ‘In case of Unintended_Acceleration the
  Engine_Control_Unit SHALL switch into Safe State [SS-01].’
  (Functional Safety Requirement, ASIL B)
```

As a result, the newly defined safe state [SS-01] and its definition were added to the LESS specification document. Also, the updated requirement [SReq-06b.1.1] was added.

4.2 Checking Consistency and Completeness of Safety and Security Requirements

For checking consistency and completeness of the requirements, we use a similar approach: We do not aim at a mathematically proven correct set of requirements, but we do support the engineer in

the formulation of an unambiguous and comprehensive specification. An example is the identification of missing steps in a chain of actions.

Example 7 (E-Gas)

> [SReq-06a.2] ‘The Engine_Control_Unit SHALL confirm undesired states of High_Driving_Torque.
> (Functional Safety Requirement, ASIL B)

Here, we find a requirement with the verb 'confirm'. Hence, the wizard asks the user:

> [SReq-06a.2] refers to the confirmation of 'undesired states of High_Driving_Torque'.
> Would you like to specify a subsequent action? [Yes/No/Later]
>
> > Y
> > [SReq-06a.2.1] In case of confirmed undesired states of High_Driving_Torque the Engine_Control_Unit SHALL switch into a safe state.
> > (Functional Safety Requirement, ASIL B)

As a result, the new requirement [SReq-06a.2.1] was added to the specification document.

In a similar way, we ask the user if multiple requirements concern the same state.

4.3 Test Case Generation

One of the main supports our methodology provides is the semi-automatic generation of test cases. As example, if a requirement mentions actions which are to be performed in certain states, the user is asked how to reach that state and how to confirm that the intended action has been performed. Here is a sample dialogue:

Example 8 (E-Gas)

> (1) [SReq-06b.1.1] ‘In case of Unintended_Acceleration the Engine_Control_Unit SHALL switch into Safe State [SS-01].’
> (Functional Safety Requirement, ASIL B)
> (2) [SS-01] ‘Injection driver of the gasoline engine is switched off’
> (Safe State)
>
> Enter Requirement ID
>
> > SReq-06b.1.1
>
> Is 'IN CASE OF Unintended_Acceleration’ a precondition or does it constrain the test input?
>
> > Test Input
>
> Describe all the ways how 'IN CASE OF Unintended_Acceleration’ can be fulfilled?
>
> > 1. Set Current_Vehicle_Acceleration > Target_Vehicle_Acceleration
>
> Are there any other preconditions for the testcase?
>
> > Y

Please enter the other preconditions below, separated by comma:

> System is running

Here is the generated testcase:

Test ID	Precondition	Test Input	Expected Behaviour
TC_001 (SReq-06b.1.1)	System is running	1. Set Current_Vehicle_Acceleration > Target_Vehicle_Acceleration	1. Engine_Control_Unit SHALL switch to Safe State [SS-01].

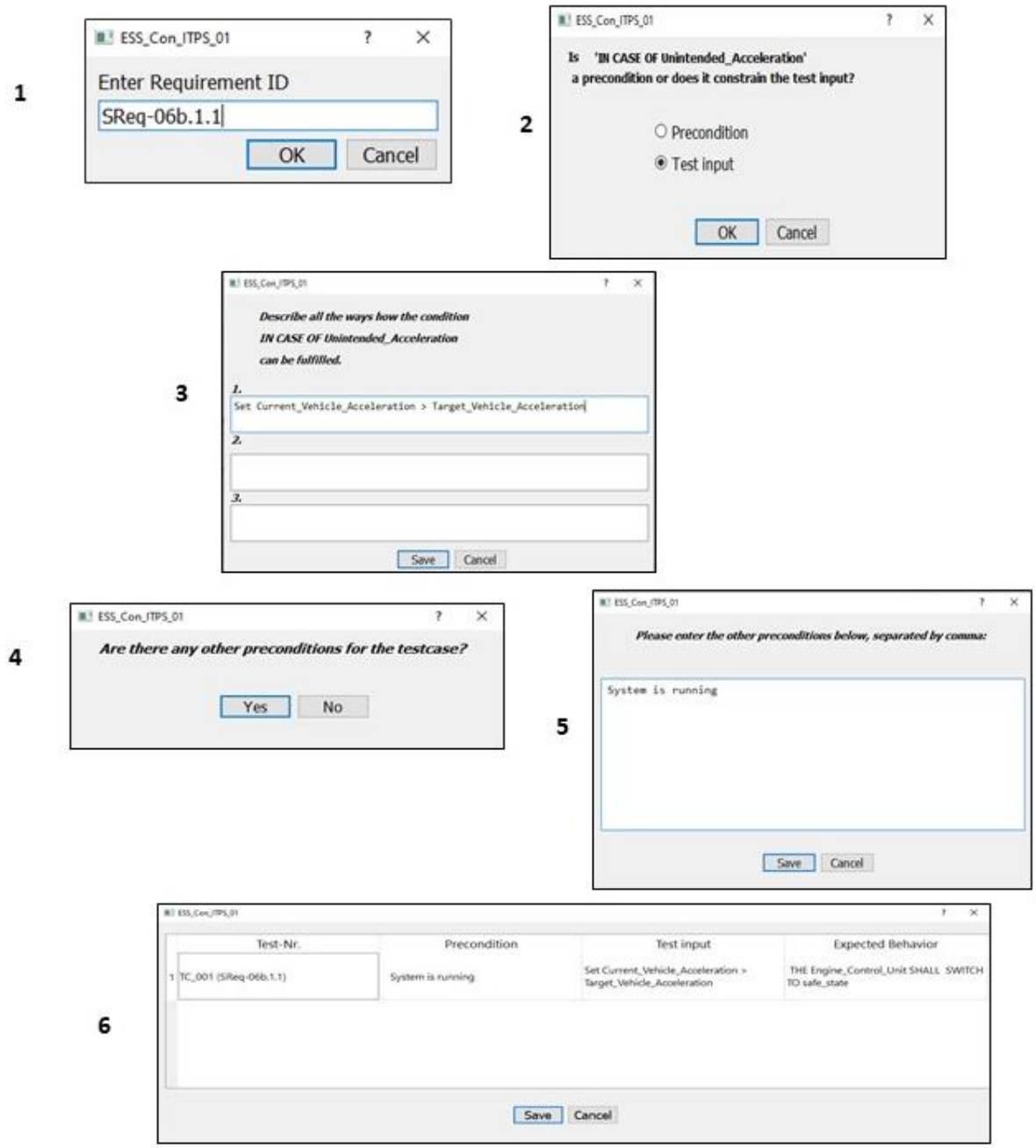

Figure 4 User interaction and test case generation workflow of the LESS tool

5 Tool Support

Within the EmbeddedSafeSec project we developed some prototypical tool support for LESS. Starting from the Xtext Java framework [25] as an Eclipse Plugin, we specified the LESS grammar using EBNF and fully-automatically generated an editor for it. This editor had features like syntax highlighting, auto-completion and error checking. Proceeding towards the processing of LESS terms and implementation of the code to access the LESS syntax tree, we faced the problem of poor documentation of Xtext and hardly understandable error messages. Therefore, we switched to the textX Python framework [26] to re-implement a simple editor for LESS, including all of the features mentioned above. Moreover, in textX we were able to realize some of the refining, checking and generation methods described in section 4, as well as the corresponding user interactions. As an example, Figure 4 shows the stepwise user interactions and the generation of a test case realized by the LESS tool prototype for Example 8 from the section 4.3.

6 Summary

In this paper, an easy-to-learn and -understand domain-specific language, called LESS, for specification of safety and security requirements of embedded software was presented. Moreover, in order to support the user by refining and analyzing requirements as well as generating test cases from them, a methodology was introduced, which is based on wizard-like conversations with the user, controlled by LESS syntax.

Within the EmbeddedSafeSec project, the LESS methodology was applied to two industrial projects from automotive and medical technology, respectively. According to this experience the authors are convinced that LESS possesses a high potential to be deployed in safety and security engineering within the embedded software industry. At the same time, one should be aware of the limitation of the LESS methodology. Firstly, it is only applicable to safety and security requirements, and not to general functional or technical requirements, which may contain certain kinds of expressions not covered by the LESS template. Secondly, the processing of the requirements formulated in LESS does not contain any semantic aspect. As a consequence, the procedures used are not fully automatic and rely highly on user interactions to achieve semantic information. Otherwise, complex mathematical/logical derivation procedures would be needed, which in turn require a complex notation and experienced users.

LESS contains also a notion for state diagrams and the corresponding methods for refinement, analysis of requirements as well as test case generation. The description of this feature would be beyond the scope of this paper.

Of course, for an industrial deployment of LESS it would be necessary to adopt and adjust the LESS template to different application fields. Last but not least, the present prototypical tool support should be developed further in order to gain more user acceptance.

Acknowledgements: The authors would like to thank Katherina Babenkova, Daniel Yermakov, Erik Haarländer, Erik Dölling and Gheorghe Celac for their valuable contributions regarding the design of the LESS DSL and the implementation of the supporting tools.

7 Bibliography

[1] *Road vehicles - Functional safety,* ISO 26262, 2018.

[2] *Road vehicles — Cybersecurity engineering,* ISO/SAE 21434, 2021.

[3] S. Curtis, "Hacker remotely crashes Jeep from 10 miles away," *The Telegraph,* 21. July 2015.

[4] "EmbeddedSafeSec project," 2022. [Online]. Available: https://embeddedsafesec.zesys.de/en/. [Accessed 30 12 2022].

[5] M. Conrad, S. Sadeghipour and H. Schlingloff, "EmbeddedSafeSec – Safety und Security Co-Engineering für eingebettete Systeme," in *SAEC Days 2022*, Munich, 2022.

[6] G. Macher, A. Höller, H. Sporer, E. Armengaud and C. Kreiner, "A Combined Safety-Hazards and Security-Threat Analysis Method for Automotive Systems," in *Proceedings of 18th Design, Automation Test in Europe Conference - DATE*, 2015.

[7] C. Schmittner, P. P. Puschner, T. Gruber and E. Schoitsch, "Security Application of Failure Mode and Effect Analysis (FMEA)," in *Proceedings of Computer Safety, Reliability, and Security, SAFECOMP 2014*, 2014.

[8] Microsoft Corporation, "The Stride Threat Model," 2009. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN. [Accessed 27. January 2023].

[9] S. Hernan, S. Lambert, T. Ostwald and A. Shostack, "Uncover Security Design Flaws Using The STRIDE Approach," *MSDN Magazine,* 2006.

[10] G. Sabaliauskaite and A. Mathur, "Aligning Cyber-Physical System Safety and Security," in *Proceedings of Complex Systems Design, Planning & Management*, 2014.

[11] I. N. Fovino and M. Masera, "Integrating Cyber Attacks within Fault Trees," *Reliability Engineering & System Safety,* vol. 94, no. 9, 2009.

[12] R. Oates, D. Foulkes, G. Herries and D. Banham, "Practical Extensions of Safety Critical Engineering Processes for Securing Industrial Control Systems," in *Proceedings of 8th IET International System Safety Conference incorporating the Cyber Security Conference*, 2013.

[13] W. Young and N. G. Leveson, "An Integrated Approach to Safety and Security Based on Systems Theory," *Communications of the ACM,* vol. 57, no. 2, 2014.

[14] I. Friedberg, K. McLaughlin, P. Smith, D. Laverty and S. Sezer, "STPA-SafeSec: Safety and Security Analysis for Cyber-Physical Systems," *Journal of Information Security and Applications,* vol. 34, no. 2, 2017.

[15] D. Pereira, C. Hirata, R. Pagliares and S. Nadjm-Tehrani, "Towards Combined Safety and Security Constraints Analysis," in *Computer Safety, Reliability, and Security: Proceedings of SAFECOMP 2017 Workshops, ASSURE, DECSoS, SASSUR, TELERISE, and TIPS*, 2017.

[16] M. Roggenbach, A. Cerone, H. Schlingloff, G. Schneider und S. Shaikh, Formal Methods for Software Engineering: Languages, Methods, Application Domains. Springer, 2022.

[17] J. R. Abrial, The B Book – Assigning Programs to Meanings, Cambridge University Press, 2005.

[18] J. Spivey, The Z Notation. A Reference Manual, 2 ed., New York: Prentice Hall, 1992.

[19] L. Lamport, Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers, Addison-Wesley Professional., 2002.

[20] T. Kuhn, "A Survey and Classification of Controlled Natural Language," *Computational Linguistics,* vol. 40, pp. 121-170, 03 2014.

[21] K. Teschner and S. Baronick, "Formal Requirements – Least heeded when most needed?" Model Engineering Solutions, 2022. [Online]. Available: https://model-engineers.com/en/academy/webinars/archive/formal-requirements-least-heeded-when-most-needed-2/. [Accessed 16 01 2023].

[22] C. Rupp, Requirements-Engineering und -Management, Munich: Hanser, 2002.

[23] P. Vallejo, R. Mazo, C. Jaramillo and J. Medina, "Towards a New Template for the Specification of Requirements in Semi-Structured Natural Language," *Journal of Software Engineering Research and Development,* vol. 8, 2 2020.

[24] EGAS Working Group, "Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units," V6.0, 2015.

[25] Eclipse Foundation, "Xtext – Language Engineering for everyone!," [Online]. Available: https://www.eclipse.org/Xtext/. [Accessed 16 01 2023].

[26] I. R. Dejanović, "textX – Domain-Specific Languages and parsers in Python made easy," [Online]. [Accessed 16 01 2023].