2015 8th International Symposium on Software and Systems Traceability

The Parameterized Safety Requirements Templates

Pablo Oliveira Antonino, Mario Trapp
Embedded Systems Division
Fraunhofer IESE
Kaiserslautern, Germany
{pablo.antonino, mario.trapp}@iese.fraunhofer.de

Paulo Barbosa, Luana Sousa
Nucleus for Strategic Health Technologies - NUTES
State University of Paraíba - UEPB
Campina Grande – PB, Brazil
{paulo.barbosa, luana.janaina}@nutes.uepb.edu.br

Abstract— **Despite imposing strict recommendations to be considered during the specification of safety requirements, standards and regulations do not provide guidance to be used throughout the creation of these artifacts. In practice, each safety requirement specification has heterogeneous structures, usually based on the experience of the engineers involved in the specification process. Consequently, it becomes difficult to ensure that the standards' recommendations were considered, such as the existence of evidences that the requirements are properly traceable to other development artifacts such as architecture and failure propagation models. To address this challenge, we defined the Parameterized Safety Requirements Templates, which is a controlled natural language based approach to support engineers in elaborating the content description of safety requirements specifications, ensuring that elements of the architectural design and of the failure propagation models are explicitly considered throughout the textual description of the safety requirements, and are therefore properly traced. The Parameterized Safety Requirements Templates have been used in different domains such as automotive, avionics, and medical devices, and have proven to be effective in improving artifact traceability. In this paper, we present their usage in the context of an industrial Automated External Defibrillator system.**

Keywords—safety requirement, architecture, failure propagation model, traceability

I. Introduction

Safety requirements should express means for avoiding, detecting, and handling potential failures to which safety-critical systems are subject [1]. Ideally, their descriptions should indicate acceptable failure mode and rates, qualitative requirements for failure modes, as well as reference elements of the architecture that address the safety requirements demands [2][3]. Some theoreticians and even some practitioners understand that, to ensure that the descriptions of safety requirements will properly express these items, it is necessary to use rigorously defined, leading-edge formal approaches and notations such as Z [4]. However, as for traditional requirements engineering activities, the vast majority of safety requirements are written in natural language [1][2][3][4][5]. The problem of using unconstrained natural languages is that their use generally results in textual specifications that, due to the lack of precision and structure, are not properly traceable to other development artifacts such as architecture and failure propagation models, are unfeasible, unnecessary, or wordy, among other things [2][6][7].

To address this challenge, we have instantiated the notion of Controlled Natural Languages [8] and created the Parameterized Safety Requirements Templates, which aim at supporting engineers with means for ensuring that traceability to architecture and to failure propagation models are present in safety requirements specifications. These guidelines are particularly important because, although we do understand that safety engineers should have freedom in specifying safety requirements with natural language, we believe that some guidance is important to ensure that elements of the architecture and of the failure propagation models are explicitly considered throughout the textual description of the safety requirements, and are therefore properly traced.

The Parameterized Safety Requirements Templates have been used in several projects in the automotive, avionics, and medical device industries in Europe and Brazil, and have proven to be effective in improving artifact traceability. In this paper, we present the use of the approach in the safety requirements specification of an Automated External Defibrillator (AED) system, which has been specified by the research institute NUTES [1] in a technology transfer collaboration process with the medical device producer company Lifemed[2].

The remainder of this paper is structured as follows: In section II, we discuss the challenges and directions regarding the specification of safety requirements with natural language. In section III, we present the state of the art on specifying safety requirements with controlled natural languages, and in section IV we present the Parameterized Safety Requirements Templates. In section V, we describe their usage in the context of an AED system and discuss the results. Finally, in section VI, we present conclusions and discuss future work.

II. Challenges and Directions on Specifying Safety Requirements with Natural Language

Standards and regulations such as ISO 26262 (Road vehicles) [9], DO-178C (Aviation) [10], and ANSI/AAMI/IEC 62304 (Medical devices) [11] describe strict recommendations to be considered during the specification of safety requirements. ISO 29148 (Systems and software engineering - Life cycle processes - Requirements engineering) [6], for instance, recommends providing evidences showing that the requirements are, among other things, fully traceable to all

[1]http://nutes.uepb.edu.br/

[2]http://www.lifemed.com.br/

978-0-7695-5593-5/15 $31.00 © 2015 IEEE
DOI 10.1109/SST.2015.12

system artifacts. Nevertheless, none of these norms specifies guidelines to be followed during the specification process. What happens in practice is that, as for traditional requirements engineering activities, the vast majority of safety requirements are written in natural language [2][3][4].

The problem of using unconstrained natural languages is that their use generally results in problematic specifications. In this regard, studies have shown that, due to the flexibility that engineers have in specifying the requirements, it is unpractical to ensure that important aspects such as traceability and consistency will be addressed in requirements written with natural language [5]. Actually, it has been shown that the more flexibility engineers have, the less precise and deterministic their requirements specifications [4]. A real example of this is presented by Mavin and Wilkinson [2], Mavin, Wilkinson, Harwood and Novak [12], and Trzakis [7], who identified several issues found in requirements specified with natural language at Rolls-Royce PLC and Intel, comprising, among other issues, missing traceability, ambiguity, vagueness, duplication, and wordiness.

In the context of safety-critical systems, the traceability aspect is particularly important when it comes to certification processes, because the documentation of traceability among safety requirements, failure propagation models, and architecture specification is mandatory [13]. However, due to the lack of guidelines and the pressure to deliver systems as soon as possible, documentation is usually neglected by companies that produce these systems, being created right before the certification processes. The consequence is that the traceability documentation is very often incomplete, wrong, and inconsistent [13].

The requirements engineering community understands that one of the means for addressing these challenges is to impose constraints on natural language in such a way that it can still be used, but with certain restrictions and guidelines [2][4][3][7][12]. The community has named this approach Controlled Natural Language [8].

A. Controlled Natural Languages

A Controlled Natural Language is a constructed (written) language based upon a natural language, restricted in terms of lexicon, syntax, and/or semantics, and has an application-specific vocabulary that can be accurately and efficiently processed by a computer, but is still expressive enough to allow natural usage [8]. In a sense, it is based on pre-formatted requirements templates that can be instantiated according to the nature and purpose of the requirements [5]. The other terms used in the literature to refer to controlled natural languages are *sublanguage, fragments of language, style guide, phraseology, controlled vocabulary,* and *constructed language* [8]. In Table I, one example of a requirements template referenced by Tommila and Pakonen [5] is shown, which illustrates what controlled natural languages are actually about.

TABLE I. REQUIREMENTS TEMPLATES EXAMPLES (ADAPTED FROM [5]).

Requirement Template	Example
The **<system function>** shall be able to **<action><entity>** at least **<quantity>** times per **<time unit>.**	***The*** missile launcher ***shall be able to*** fire missiles ***at least*** 15 ***times per*** minute

Requirements templates are also referenced in the literature as Requirements Boilerplates, which, like the term templates, refers to text portions that can be reused without being modified too much compared to the original [5][14].

B. Aspects to Consider when Specifying Safety Requirements with Controlled Natural Languages

As previously mentioned, safety requirements should be precisely traceable to (i) what motivates their existence, which, in turn, are failures described in failure propagation models such as Failure Mode and Effect Analysis (FMEA) [15], Markov Chains [16], and Component Fault Trees [17], and (ii) the system elements that address them, which, in turn, are elements described in multiple architecture views [18]. As a first step towards achieving this goal, we presented the Safety Requirements Decomposition Pattern in a previous work of ours, which is a model-based structural guideline to be followed throughout the decomposition of safety requirements [19]. In short, the Safety Requirements Decomposition Pattern suggests that it is necessary to specify safety requirements for detecting and containing failures associated with each abstraction level of the system: from the high-level functional specification to the software and hardware elements that realize them. In this regard, we understand that a controlled natural language should offer means for specifying safety requirements that explicitly reference failure propagation models and architectural elements from multiple hierarchical levels of the system decomposition.

We understand that the controlled natural language should also comply with best practices described in the state of the art and in regulation recommendations, such as those from ISO 29148 [6], which, among other things, recommends: (i) “It is best to avoid using the term ‘must', due to potential misinterpretation as a requirement”; (ii) “Use positive statements and avoid negative requirements such as ‘shall not'”; and (iii) “Use active voice: avoid using passive voice, such as 'shall be able to select'”.

III. LITERATURE REVIEW

We have investigated the existing approaches in the literature that could support us in specifying safety requirements such that the aspects described in Section II.B are met, and they are described in the remainder of this section.

In 2014, T. Kuhn [8] elaborated a survey describing one hundred English-based controlled natural languages. Out of all the languages analyzed by Kuhn, only Attempto could have contributed to the definition of the Parameterized Safety Requirements Templates described in this paper, due to its support for elaborating complex noun phrases, plurals, anaphoric references, subordinated clauses, modality, and questions. However, it lacks on precise support to accurately support the elaboration of textual constructs that explicitly refer to failure propagation models or to elements of the architecture specifications.

The CESAR Requirement Specification Language [20] comprises twelve templates that capture probability of failure occurrence, failure lists, among others. However, the CESAR templates do not offer the necessary means to jointly specify failures and the elements that are subject to the failure, nor the containment and detection mechanisms. For instance, their failure detection requirements pattern expresses that a certain failure should be detected, but do not specify which architectural element should perform the detection. In this regard, the parameterized safety requirements templates presented in this paper evolve the CESAR safety requirements templates by offering means to precisely indicate in the specification of safety requirements not only the the failure per se, but its origin and the mitigation mechanisms expressed in detection and containment requirements.

The EARS (Easy Approach to Requirements Syntax) [2][4][7] is a broadly accepted controlled language that was developed at Rolls-Royce Control Systems, and that has been used in the context of safety-critical systems like aero-engine controls. One aspect that is also claimed to address is traceability. However, their focus is on tracing lower-level requirements to high-level requirements. The EARS traceability means to trace the requirements and architecture elements are not enough to trace elements from multiple architecture views. Moreover, the EARS templates do not provide means at all to reference failure models, neither to explicitly specify detection and containment mechanisms. In a nutshell, EARS is an efficient and practical way to enhance the writing of high-level stakeholder requirements, but lacks on means to proper specify safety requirements.

Last, the Requirements templates for Safety-related I&C systems [5] offer a good basis for referencing failure propagation models and architecture elements from multiple views. However, as the name suggests, they were specifically designed for Instrumentation and Control systems, and instantiating them to other classes of systems might require complete restructuring of the language.

In a nutshell, despite the extensive work on controlled natural languages, none of the existing approaches can be used directly to address our needs as described in section II.B.

IV. The Parameterized Safety Requirements Templates

In this section, we present the Parameterized Safety Requirements Templates. For the sake of systematicity, we first present the template for the *Top-Level Safety Requirement* [19], followed by the templates for the safety requirements at the functional level, and finally by the templates for the safety requirements at the technical level, which comprises software and hardware artifacts.

A. Parameterized Template for Top-Level Safety Requirements

The *Top-Level Safety Requirement* aims at providing a high-level description of the generic safety needs to address the failures identified during safety analysis and documented in failure propagation models [19]. The *Top-Level Safety Requirement* is called differently depending on the domain. For instance, in the road vehicles domain, it is called Safety Goal [9]. The parameterized template to be considered when specifying top-level safety requirements is shown in Fig. 1.

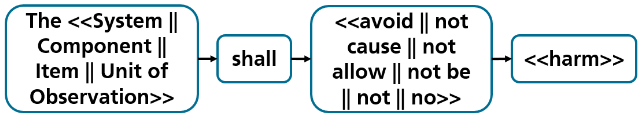


Fig. 1.Parameterized template for Top-Level Safety Requirements.

B. Parameterized Template for Functional Safety Requirements

At the functional level, we specified parameterized templates for *Functional Detection Requirements* and for *Functional Containment Requirements*. We understand that the specification of these two types of safety requirements should be textually constrained because they express means for taking the functional architecture from a state *containing errors and faults* to a state *without undetected errors and without faults.* Therefore, constrained guidelines will coerce engineers to write these safety requirements in such a way that the specifications do indeed express detection and containment measures. The templates for the Functional Detection Requirements and for the Functional Containment Requirements are shown in Fig. 2 and Fig. 3, respectively.

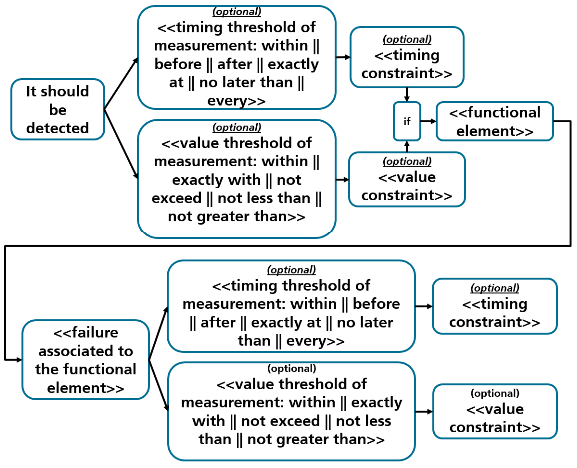

It should be detected
(optional)
<<timing threshold of measurement: within || before || after || exactly at || no later than || every>>
(optional)
<<timing constraint>>
(optional)
<<value threshold of measurement: within || exactly with || not exceed || not less than || not greater than>>
(optional)
<<value constraint>>
if
<<functional element>>
<<failure associated to the functional element>>
(optional)
<<timing threshold of measurement: within || before || after || exactly at || no later than || every>>
(optional)
<<timing constraint>>
(optional)
<<value threshold of measurement: within || exactly with || not exceed || not less than || not greater than>>
(optional)
<<value constraint>>

Fig. 2. Parameterized template for Functional Detection Requirements.

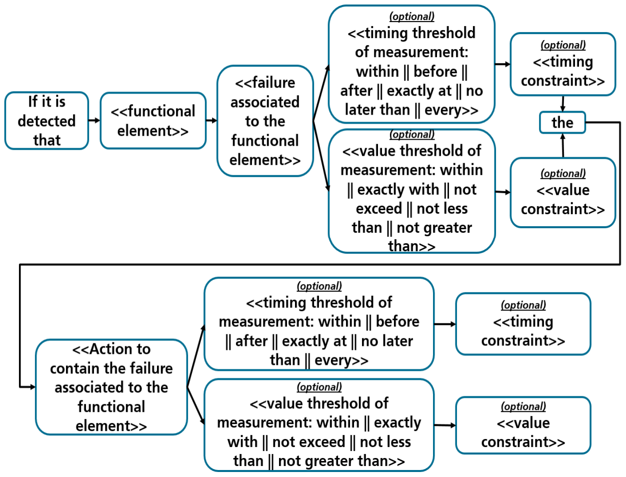

If it is detected that
<<functional element>>
<<failure associated to the functional element>>
(optional)
<<timing threshold of measurement: within || before || after || exactly at || no later than || every>>
(optional)
<<timing constraint>>
(optional)
<<value threshold of measurement: within || exactly with || not exceed || not less than || not greater than>>
(optional)
<<value constraint>>
the
<<Action to contain the failure associated to the functional element>>
(optional)
<<timing threshold of measurement: within || before || after || exactly at || no later than || every>>
(optional)
<<timing constraint>>
(optional)
<<value threshold of measurement: within || exactly with || not exceed || not less than || not greater than>>
(optional)
<<value constraint>>

Fig. 3. Parameterized template for Functional Containment Requirements.

C. Parameterized Template for Technical Safety Requirements

The description of the safety requirements at the technical level should correspond to a realization of the safety requirements at the functional level considering software and hardware elements. The first template presented for this level is the one for the Technical Safety Requirement element, as shown in Fig. 4.

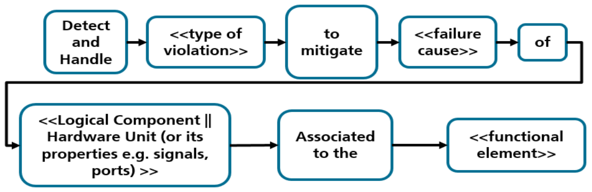


Fig. 4. Parameterized template for Technical Safety Requirements.

Due to the importance of detection and containment requirements and the consequent need to constrain their textual descriptions as discussed in section IV-B, we specified parameterized templates for the *Technical Detection* and *Containment Requirements*, which are shown in Fig. 5 and Fig. 6, respectively.

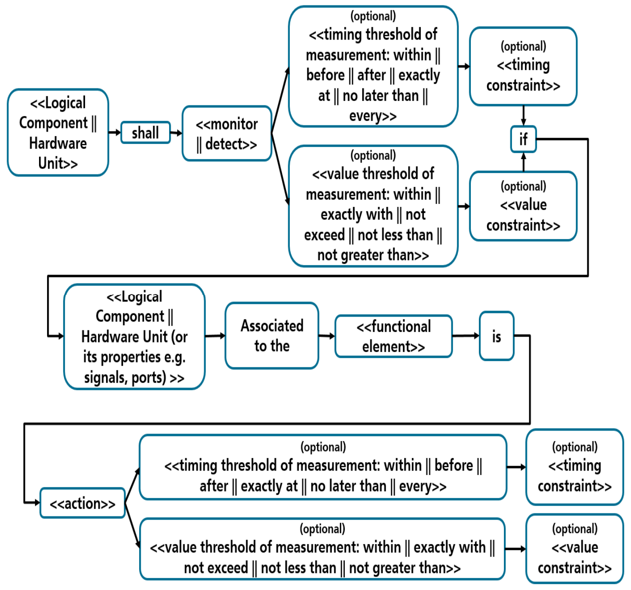

<<Logical Component || Hardware Unit>>
shall
<<monitor || detect>>
(optional) <<timing threshold of measurement: within || before || after || exactly at || no later than || every>>
(optional) <<timing constraint>>
if
(optional) <<value threshold of measurement: within || exactly with || not exceed || not less than || not greater than>>
(optional) <<value constraint>>
<<Logical Component || Hardware Unit (or its properties e.g. signals, ports) >>
Associated to the
<<functional element>>
is
<<action>>
(optional) <<timing threshold of measurement: within || before || after || exactly at || no later than || every>>
(optional) <<timing constraint>>
(optional) <<value threshold of measurement: within || exactly with || not exceed || not less than || not greater than>>
(optional) <<value constraint>>

Fig. 5. Parameterized template for Technical Detection Requirements.

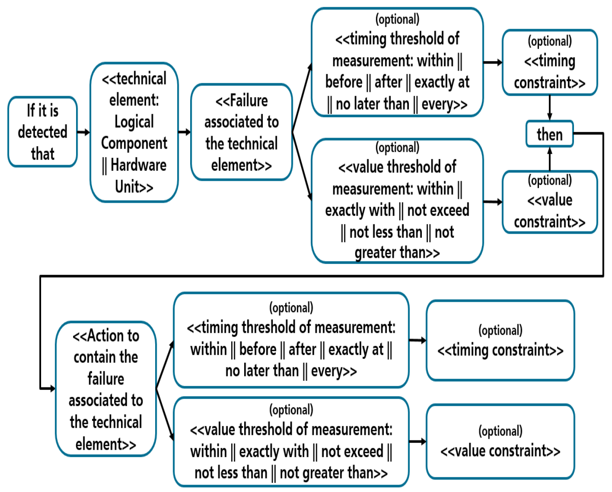

If it is detected that
<<technical element: Logical Component || Hardware Unit>>
<<Failure associated to the technical element>>
(optional) <<timing threshold of measurement: within || before || after || exactly at || no later than || every>>
(optional) <<timing constraint>>
then
(optional) <<value threshold of measurement: within || exactly with || not exceed || not less than || not greater than>>
(optional) <<value constraint>>
<<Action to contain the failure associated to the technical element>>
(optional) <<timing threshold of measurement: within || before || after || exactly at || no later than || every>>
(optional) <<timing constraint>>
(optional) <<value threshold of measurement: within || exactly with || not exceed || not less than || not greater than>>
(optional) <<value constraint>>

Fig. 6. Parameterized template for Technical Containment Requirements.

V. Using Parameterized Safety Requirements Templates for Specifying the Safety Requirements of an Automated External Defibrillator

AED is a medical device used for ventricular fibrillation/tachycardia treatment, which are cardiac arrhythmias with the highest incidences of fatal cases. The AED is operated by a rescuer who operates the device in order to deliver controlled energy to the patient's chest. The amount of energy delivered is based on an automated analysis of the Electrocardiography (ECG), which is also performed by the AED. Delays in the application of the defibrillator shock are safety-critical because every minute without the application of the shock implies 7% to 10% decrease of the chance of survival. In this regard, it is important to ensure optimized recharging to minimize the intervals of discharge.

We explored the AED that has been specified by NUTES in a technological transfer process with Lifemed to demonstrate how the Parameterized Safety Requirements Templates contributed positively to improving traceability within the engineering processes of the AED. NUTES stands for Nucleus for Strategic Health Technologies and is an initiative by the Brazilian Health Ministry aimed at promoting the technological development of medical devices.

This section is organized as follows: In section V-A, we present the preliminary specification of the AED, describing the initial architecture, the safety analysis, and the safety requirements. In section V-B, we present the safety requirements specification restructured according to the Safety Requirements Decomposition Pattern and how the preliminary architecture was modified to meet the safety needs. Finally, in section V-C we discuss the results.

A. Preliminary Specification of the Automated External Defibrillator

Fig. 7 depicts the context diagram for the AED system. This diagram aims at showing each external entity, the main functional units, and their interactions. The main input variable to be received is the cardiac pulse of the patient. The *Signal Analyzer* employs sophisticated algorithms for detecting the signal complexity and determines whether a cardiac arrest has occurred. If this is the case, the *Shock Generator* provides controlled energy in the Biphasic Truncated Exponential waveform to the patient's chest through the cardiac pads.

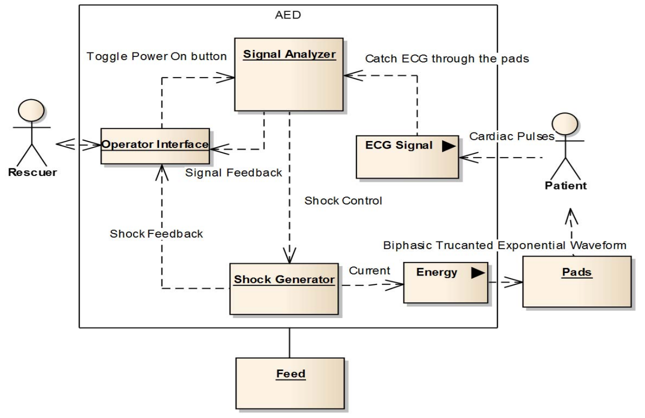


Fig. 7. Context diagram of the AED system.

Fig. *8* depicts the main AED use cases, where the normal operation of the AED includes both configuring the AED and obtaining the appropriate energy. Moreover, the shock operation can be aborted and the device can perform a self-test, most likely while booting.

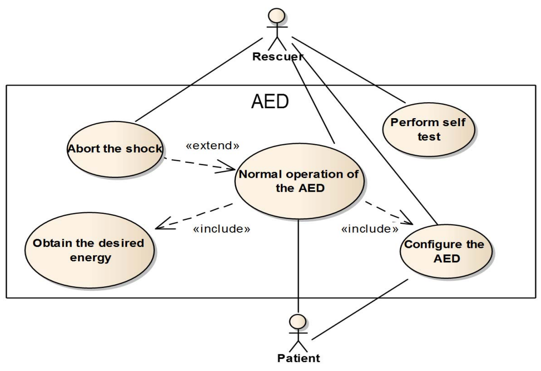
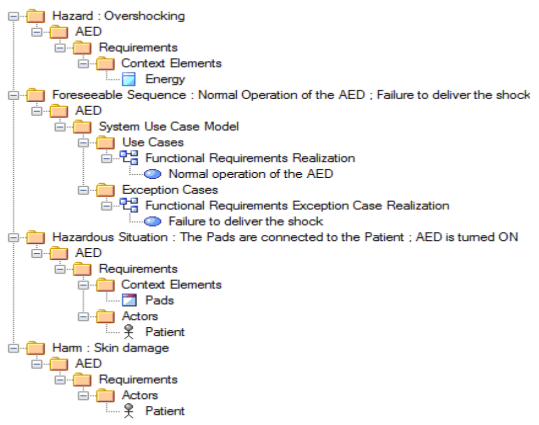


Fig. 8. Main AED Use Cases impacting the hazard of overshocking.

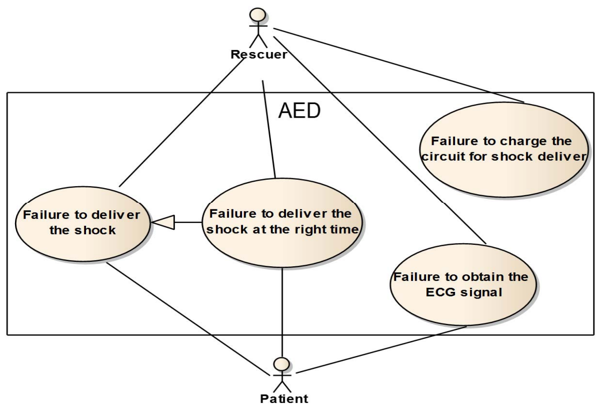
In order to identify potential safety-critical failures of the AED, we conducted quantitative and qualitative safety analysis considering heterogeneous artifacts of the system, such as those depicted in Fig. 7 and

Fig. 8. The analysis was supported by an Enterprise Architect[3] based solution developed at NUTES, which offers safety analysis support customized for meeting the demands of ISO 14971 [21] and its specific technical report IEC/TR 80002 [22]. An excerpt of the analysis is depicted in Fig. 9, where the hazard of *overshocking* can be seen, which is associated with the use case *Normal operation of the AED* use case (cf. Fig. 8); the main parameter to be controlled regarding this hazard is *energy*. A more detailed overview of the abnormal occurrence of the *Normal operation of the AED* (cf. Fig. 8) use case is depicted in Fig. 10, in which the *Failure to deliver the shock* use case and its implications can be seen.



```
Hazard : Overshocking
    AED
        Requirements
            Context Elements
                Energy
Foreseeable Sequence : Normal Operation of the AED ; Failure to deliver the shock
    AED
        System Use Case Model
            Use Cases
                Functional Requirements Realization
                    Normal operation of the AED
            Exception Cases
                Functional Requirements Exception Case Realization
                    Failure to deliver the shock
Hazardous Situation : The Pads are connected to the Patient ; AED is turned ON
    AED
        Requirements
            Context Elements
                Pads
            Actors
                Patient
Harm : Skin damage
    AED
        Requirements
            Actors
                Patient
```

Fig. 9. Enterprise Architect add-in for tracing hazard analysis to architectural elements.



Rescuer
AED
Failure to charge the circuit for shock deliver
Failure to deliver the shock
Failure to deliver the shock at the right time
Failure to obtain the ECG signal
Patient

Fig. 10. Exception Use Cases designed for improving a system's reliability and which are related to the hazard of overshocking

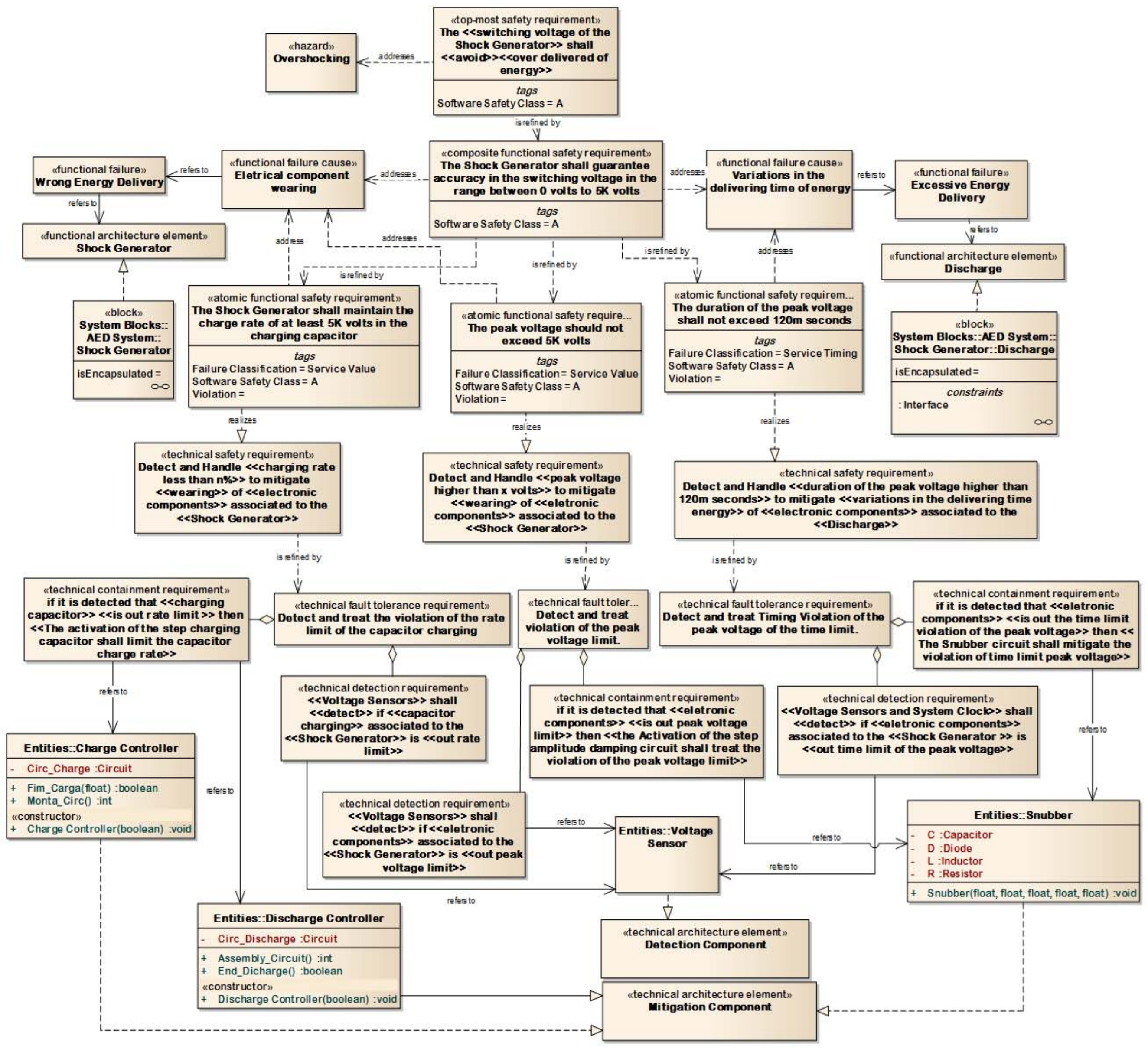
B. Safety Requirements Specification for Addressing the Hazard of "Overshocking"

We specified safety requirements for mitigating the hazard of *overshocking,* and the result is summarized in Fig. 11. For hierarchically decomposing the safety requirements we used the Safety Requirements Decomposition Pattern [19] and used the Parameterized Safety Requirements Templates presented in this paper to elaborate the contents of each decomposition unit.

One of the measures for mitigating AED overshocking is to ensure that the switching voltage of the shock generator (cf. Fig. 7) will not deliver energy above the specified value. This aspect is represented by the top-most safety requirements element in Fig. 11. The main causes that might lead to the occurrence of this hazard are (i) wear of electronic components or (ii) an unexpected loss of control from the software towards unexpected values. This motivates new fine-grained specifications such as charging rates for capacitors, specific voltage peak values, or specific frequencies via semiconductor switching. Therefore, three technical safety requirements were derived and matched perfectly to those defined in the Parameterized Safety Requirements Templates. The first two cases, the leftmost ones, refer to the architectural element Shock Generator module, while the last one, the rightmost one, refers to the architectural element Discharge module. This parameterization makes it easier to clearly define the goal of each subtree specification.

Going towards fine-grained specifications, the next refinement are fault tolerance requirements such as indication of wear of components; warning that maintenance and repair should be provided; coupling of new amortization circuits in order to deal with unexpected peak values; or coupling of new circuits for the management of periods to avoid extrapolation of time limits for issuing peak tensions. This generates technical containment requirements and technical detection requirements, which are also defined according to the parameterized templates. Finally, the last layer of specification will require specific sensors and actuators for efficient detections and actions over all hazardous conditions.

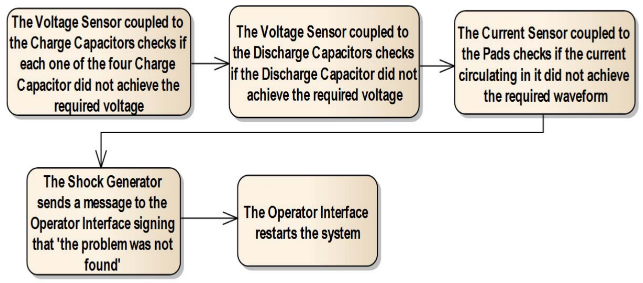
3 www.sparxsystems.com



«hazard» Overshocking
addresses
«top-most safety requirement» The <<switching voltage of the Shock Generator>> shall <<avoid>><<over delivered of energy>>
tags Software Safety Class = A
is refined by
«composite functional safety requirement» The Shock Generator shall guarantee accuracy in the switching voltage in the range between 0 volts to 5K volts
tags Software Safety Class = A
«functional failure» Wrong Energy Delivery
«functional failure cause» Eletrical component wearing
«functional failure cause» Variations in the delivering time of energy
«functional failure» Excessive Energy Delivery
«functional architecture element» Shock Generator
«functional architecture element» Discharge
«block» System Blocks::AED System::Shock Generator
isEncapsulated =
«block» System Blocks::AED System::Shock Generator::Discharge
isEncapsulated = constraints : Interface
«atomic functional safety requirement» The Shock Generator shall maintain the charge rate of at least 5K volts in the charging capacitor
tags Failure Classification = Service Value Software Safety Class = A Violation =
«atomic functional safety require... The peak voltage should not exceed 5K volts
tags Failure Classification = Service Value Software Safety Class = A Violation =
«atomic functional safety requirem... The duration of the peak voltage shall not exceed 120m seconds
tags Failure Classification = Service Timing Software Safety Class = A Violation =
refers to
realizes
«technical safety requirement» Detect and Handle <<charging rate less than n%>> to mitigate <<wearing>> of <<electronic components>> associated to the <<Shock Generator>>
«technical safety requirement» Detect and Handle <<peak voltage higher than x volts>> to mitigate <<wearing> of <<eletronic components>> associated to the <<Shock Generator>>
«technical safety requirement» Detect and Handle <<duration of the peak voltage higher than 120m seconds>> to mitigate <<variations in the delivering time energy>> of <<electronic components>> associated to the <<Discharge>>
«technical containment requirement» if it is detected that <<charging capacitor>> <<is out rate limit >> then <<The activation of the step charging capacitor shall limit the capacitor charge rate>>
«technical fault tolerance requirement» Detect and treat the violation of the rate limit of the capacitor charging
«technical fault toler... Detect and treat violation of the peak voltage limit.
«technical fault tolerance requirement» Detect and treat Timing Violation of the peak voltage of the time limit.
«technical containment requirement» if it is detected that <<eletronic components>> <<is out the time limit violation of the peak voltage>> then << The Snubber circuit shall mitigate the violation of time limit peak voltage>>
«technical detection requirement» <<Voltage Sensors>> shall <<detect>> if <<capacitor charging>> associated to the <<Shock Generator>> is <<out rate limit>>
«technical containment requirement» if it is detected that <<eletronic components>> <<is out peak voltage limit>> then <<the Activation of the step amplitude damping circuit shall treat the violation of the peak voltage limit>>
«technical detection requirement» <<Voltage Sensors and System Clock>> shall <<detect>> if <<eletronic components>> associated to the <<Shock Generator >> is <<out time limit of the peak voltage>>
«technical detection requirement» <<Voltage Sensors>> shall <<detect>> if <<eletronic components>> associated to the <<Shock Generator>> is <<out peak voltage limit>>
Entities::Charge Controller
- Circ_Charge :Circuit
+ Fim_Carga(float) :boolean
+ Monta_Circ() :int
«constructor»
+ Charge Controller(boolean) :void
Entities::Voltage Sensor
Entities::Snubber
- C :Capacitor
- D :Diode
- L :Inductor
- R :Resistor
+ Snubber(float, float, float, float, float) :void
Entities::Discharge Controller
- Circ_Discharge :Circuit
+ Assembly_Circuit() :int
+ End_Dicharge() :boolean
«constructor»
+ Discharge Controller(boolean) :void
«technical architecture element» Detection Component
«technical architecture element» Mitigation Component

Fig. 11. AED safety requirements specified using the Parameterized Safety Requirements Templates for mitigating the hazard of "overshocking"

The mitigation strategy for the *overshocking* hazard is further described in Fig. 12, which depicts an excerpt of the activity diagram that summarizes the alternative flow to deliver the shock with mitigation procedures that interact with software and hardware components identified during the specification of the safety requirements using the Parameterized Safety Requirements Templates, and which were modeled and analyzed during the analysis and design phase using tools such as Enterprise Architect, Matlab/Simulink[4], and ISOGRAPH/Reliability Workbench[5].



The Voltage Sensor coupled to the Charge Capacitors checks if each one of the four Charge Capacitor did not achieve the required voltage
The Voltage Sensor coupled to the Discharge Capacitors checks if the Discharge Capacitor did not achieve the required voltage
The Current Sensor coupled to the Pads checks if the current circulating in it did not achieve the required waveform
The Shock Generator sends a message to the Operator Interface signing that 'the problem was not found'
The Operator Interface restarts the system

Fig. 12. Excerpt of activity diagram that summarizes the mitigation alternative flow "Failure to deliver the shock"

4 www.mathworks.com

5 www.isograph.com

C. Brief Discussion

The current version of the Parameterized Safety Requirements Templates presented in this paper is the result of intensive use in the road vehicles, avionics, and medical devices domains. In the road vehicles domain, it has been used for specifying the safety requirements of a power sliding door system, of an Adaptive Driving Lights system, and of an E-Drive system. In the avionics domain, it has been used to specify a collision avoidance system of an aircraft. In the medical devices domain, in addition to being used for the cardiac defibrillator, it has been used in the development of an infusion pump and vital sign monitors.

The NUTES scientists have reported that the Parameterized Safety Requirements Templates are an example of a *de facto* contribution to reduce risks caused by erroneous and incomplete specifications. Moreover, it enables precise and traceable specifications of safety requirements with considerably reduced costs compared to very sophisticated techniques. This is an important aspect, which makes its acceptance by medical device manufacturers a more realistic prospect. As Daniel Jackson pointed out: "*The tendency (in Z especially) to see a specification language as a general mathematical notation is surely a mistake, since such generality can only come at the expense of analysis (and, moreover, at the expense of the language's suitability for its most common applications)*"[23].

Indeed, we understand that the philosophy of aligning the effect that Parameterized Safety Requirements Templates produce with the necessary effort to use it was a key factor for its widespread acceptance in multiple industries and domains.

VI. Conclusions and Future Work

In this paper we presented the Parameterized Safety Requirements Templates, which aims at supporting engineers with means to specify safety requirements with controlled natural languages to properly trace them to failure propagation models and architecture.

We presented how the approach has been used in the development context of an Automated External Defibrillator (AED) system, and discussed how it enables precise and traceable specifications of safety requirements with considerably reduced costs compared to very sophisticated techniques.

As future work we intend to refine the Parameterized Safety Requirements Templates so other safety engineering artifacts like safety cases can be referenced from the templates.

Acknowledgment

This work is supported by the Fraunhofer Innovation Cluster Digitale Nutzfahrzeugtechnologie. We would also like to thank Sonnhild Namingha for proofreading, to the head of NUTES Prof. Dr. Misael Morais, to Raphael Mendonca for the tool support implementation, and to the Prof. Dr. Edmar Gurjão and Yang Medeiros for the valuable insights on the electronic aspects of the AED.

VII. References

[1] J. Hatcliff, A. Wassyng, T. Kelly, C. Comar and P. Jones, "Certifiably safe software-dependent systems: challenges and directions," in *Future of Software Engineering (FOSE 2014)*, Hyderabad, India, 2014.

[2] A. Mavin and P. Wilkinson, "Big Ears (The Return of "Easy Approach to Requirements Engineering")," in 18th IEEE International Requirements Engineering Conference (RE 2010), Sydney, New South Wales, Australia, 2010.

[3] K. Allenby and T. Kelly, "Deriving safety requirements using scenarios," in Fifth IEEE International Symposium on Requirements Engineering, Toronto, Canada, 2001.

[4] A. Mavin, "Listen, Then Use EARS," *IEEE Software,* vol. 29, no. 2, pp. 17,18, 2012.

[5] T. Tommila and A. Pakonen, "Controlled natural language requirements in the design and analysis of safety critical I&C Systems," SAFIR2014 Reference group 2, Espoo, Finland, 2014.

[6] International Organization for Standardization, "ISO/IEC/IEEE 29148:2011 Systems and software engineering - Life cycle processes - Requirements engineering," IEEE, 2011.

[7] J. Terzakis, "EARS: The Easy Approach to Requirements Syntax," in *Tutorial at the Eighth International Multi-Conference on Computing in the Global Information Technology*, Nice, France, 2013.

[8] T. Kuhn, "A Survey and Classification of Controlled Natural Languages," in *Computational Linguistics 40(1)*, 2014.

[9] International Organization for Standardization, "ISO/DIS 26262 - Road Vehicles – Functional Safety," Technical Committee 22 (ISO/TC 22), Geneva, Switzerland, 2011.

[10] DO-178C/ED-12C, "Software Considerations in Airborne Systems and Equipment," RTCA, 2011.

[11] ANSI/AAMI/IEC 62304:2006, "Medical Device Software—Software Life Cycle," Assoc. Advancement Medical Instrumentation, 2006.

[12] A. Mavin, P. Wilkinson, A. Harwood and M. Novak, "Easy Approach to Requirements Syntax (EARS)," in *17th IEEE International Requirements Engineering Conference (RE '09)*, Atlanta, Georgia, USA, 2009.

[13] P. Maeder, P. L. Jones, Y. Zhang and J. Cleland-Huang, "Strategic traceability for safety-critical projects," *IEEE Software,* vol. 30, no. 3, pp. 58-68, 2013.

[14] E. Hull, K. Jackson and J. Dick, Requirements Engineering, SpringerVerlag, 2004.

[15] J. J. Stadler and N. J. Seidl, "Software failure modes and effects analysis," in *Annual Reliability and Maintainability Symposium (RAMS)*, 2013.

[16] G. Norman, D. Parker, and J. Sproston. 2013. Model checking for probabilistic timed automata. *Form. Methods Syst. Des.* 43, 2 (October 2013), 164-190.

[17] D. Domis and M. Trapp, "Component-Based Abstraction in Fault Tree Analysis," in *28th International Conference on Computer Safety, Reliability, and Security*, Hamburg, Germany, 2009.

[18] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers and R. Little, Documenting Software Architectures: Views and Beyond, Pearson Education, 2002.

[19] T. Kuhn, P. O. Antonino, Model Driven Development of Embedded Systems, Proceedings of the Embedded Software Engineering Congress 2014, Sindelfingen, Germany, 2014.

[20] CESAR 2010. Definition and exemplification of RSL and RMM. Publication D_SP2_R2.1_M1 of the CESAR project, http://www.cesarproject.eu/, 188 p.

[21] International Organization for Standardization, "ISO 14971:2007 Medical devices -- Application of risk management to medical devices. IEEE, 2007

[22] International Organization for Standardization, IEC/TR 80002-1:2009, Medical device software -- Part 1: Guidance on the application of ISO 14971 to medical device software.

[23] D. Jackson and J. Wing, "Lightweight Formal Methods," IEEE Computer, April 1996, pp. 21-22.