Validation of Self-Adaptive Systems' Safety Requirements at Design Time

Rasha Abu Qasem
Chair of Software Engineering: Dependability
University of Kaiserslautern, Germany
email: abuqasem@cs.uni-kl.de

Peter Liggesmeyer
Chair of Software Engineering: Dependability
University of Kaiserslautern, Germany
email: liggesmeyer@cs.uni-kl.de

***Abstract*—Self-Adaptive Systems (SAS) are becoming more evident in our lives. By definition, these systems are supposed to adapt to changes in their context without human interference. It is essential that they are trusted to autonomously perform critical tasks in changing environmental conditions. That makes assuring their safety a vital task, and because of their complexity, a challenging one. In this work, our goal is to identify potential safety vulnerabilities of SAS at design time. We start by addressing safety requirements of SAS and transferring their natural language description to a guided template then to a formal description. After that, we aim to automatically generate adequate test cases. At the same time, we build the initial models of the system and with the help of a powerful simulator, we run the generated test cases against the system models. This way, we can validate the safety requirements of SAS at design time.**

Keywords*— *Self-Adaptive Systems; Safety Requirements; Test cases; Auto-generation; Design time

I. INTRODUCTION

Self-adaptive systems usually have high levels of automation which present new risks to the surroundings. To minimize the risk to a residual value, we came up with a new approach to represent and test safety requirements of SAS. Knowing that, the bigger the gap between error introduction and error detection, the bigger the cost of development. We designed our approach to validate SAS safety requirements at design time, which will facilitate early detection of failures in the system and subsequently reduce the time and effort of the development.

We start by requirements elicitation process of SAS, which is still not a solved problem because of the uncertain behavior of these systems. Although there are some proposed solutions for the requirements problem of SAS as in [1] [2] [3], we are convinced there is no solution that specifically targets the representation of safety requirements of SAS. That is why we start by investigating how safety requirements are managed in the first place.

The Parametrized Safety Requirements Templates (PSRTs) defined by Oliveira in [4] can be used as the basis of safety requirements elicitation of SAS. PSRTs are used to check the safety requirements for conflicts and completeness by translating the natural language description to a controlled structured natural language.

We propose a possible representation of the safety requirements of SAS by extending PSRTs to Extended PSRTs (E-PSRTs). PSRTs already focus on how to reflect the safety analysis results and failure propagation models on the system's requirements and architecture. By extending PSRTs, we adjust the PSRTs templates to incorporate the managing and control requirements, the environment's conditions and the expected behavior of SAS. We will also extend the definition of safety requirements in the PSRTs to elaborate the concepts of guaranteed safety and demanded safety requirements form the conditional safety certificates (ConSerts) introduced in [5].

Creating E-PSRTs would provide a promising structure to manage safety requirements of SAS and subsequently facilitate their validation. It would also give a clear insight on how the concepts in ConSerts can be exhibited in the requirements engineering phase of SAS. That summarizes our first contribution in this work.

Later on, we define a domain specific language rules to implement the E-PSRTs templates into a formal structure. Then we use this structure to auto-generate test cases by injecting faults to the normal or expected behavior of the system. The auto-generated test cases are used besides the test cases designed by domain experts to validate the safety requirements of the system.

Finally, we run the all the test cases against the preliminary system models in a simulation environment to evaluate and check the validity and completeness of the safety requirements of SAS before the actual development. We carefully monitor the system response in risky situations and its adherence to the defined guards in safety requirements.

This would provide an early estimation of the SAS behavior at design time. If a risky or unsafe behavior is identified, we can easily trace the architecture elements, safety requirements as well as the failure propagation model to reassess them and perform the needed changes. This would be our second contribution in this work.

The remainder of this paper is structured as follows: In Section II, we discuss the problem we are targeting and identify the gaps in literature. In Section III we present our proposed approach to validate the safety requirements of SAS at design time. In Section IV, we provide the realization and validation plan of our approach with an introduction of the chosen case study. Finally, in Section V, we conclude and discuss future work.

 ISBN: 978-1-61208-781-8

II. MOTIVATION - PROBLEM

The motivation behind our approach is to find a rigorous solution to solve problems we have located through practise and research gaps we have identified in the literature. Mainly, we can classify our motivation into three categories, knowing that, more specific problems can arise in these categories as the research progresses.

A. Specification of Safety Requirements of SAS

There are several attempts of targeting uncertainty in SAS requirements, for example, in [3] probabilistic and fuzzy relaxation have been used to represent the adaptive properties of SAS's requirements. Different types of SAS requirements have been distinguished namely, monitoring, control and evolution requirements. The concept of the Configurable Specification (CP) has been introduced. CP is a set of interrelated configurations where the system can start from a configuration in which the domain assumptions are consistent with the system environment. This concept sounds a promising start to solve the problem of requirements engineering of SAS. However, the safety requirements are not addressed at all, and even the non-functional requirements are vaguely mentioned.

Another try to tackled uncertainty of SAS is with RELAX [1], which is a requirements language for SAS that explicitly addresses uncertainty inherent in adaptive systems. Its formal semantic is represented in terms of fuzzy logic, thus it enables a rigorous treatment of requirements that include uncertainty. Regarding non-functional requirements, they are dealt with in a conventional way, meaning they have their quantification of the minimum acceptable levels and are not changed by RELAX process. RELAX doesn't provides any specifics about safety requirements.

Other attempts, like in [2][6][7], focus mainly on the process of requirements elicitation and/or management of SAS without giving enough focus on the overall safety of the system or the safety requirements themselves.

On another scope, we know that the lack of guidance on how to specify safety requirements that are properly traceable to the architecture design and to failure propagation models is one of the main reasons for their incompleteness and inconsistency, which turns out to be a root cause of safety incidents [4]. That implies the importance of specifying traceable and consistence safety requirements of a system.

The fact that, to our best knowledge, there exists no clear methods neither recommendations on how to elicit or manage safety requirements of SAS, is a big drive behind our approach.

B. Validation of Safety Requirements of SAS at Design time

It has become a rule of thumb in software development, that the earlier the error is detected the less impact it has on the development's cost and effort. That is why we are targeting in our approach the safety requirements of SAS during the system design phase. By using a proper simulator to run the system models against the test cases, we can validate the safety requirements of the SAS at early stages and we can trace errors and locate potential design flows before the actual implementation takes place.

C. Auto-generation of Test Cases to verify the Safety of the SAS

Documented test cases are essential for testing large and complex systems such as SAS or the embedded systems in the automotive domain. ISO-26262 [8], an automotive safety standard, states that all system requirements should be properly tested by corresponding system test cases. Usually test cases are derived manually from the textual requirements, that is why it is a time consuming and error-pron process. Automatic test generation proved to be a powerful approach to reduce the cost of testing as well as to assure the requirements' coverage.

The benefits of automatic test generation are widely acknowledged today and there are many proposed approaches to conduct it in the literature. For instance, in [9] and [10] they proposed approaches to auto-generate UML behavioral diagrams such as, activity diagrams and sequence diagrams from use case descriptions. The generated models can be used to auto-generate test cases. However, the generated test cases are not executable and need manual intervention.

There are some other approaches which generate executable test cases such as [10] and [11]. They require that the requirements specifications are written according to a Controlled Natural Language (CNL). The input specifications are translated into formal specifications, which are later used to automatically generate test input data (e.g., using constraint solving). The problem with these approaches is that the CNL language supported by them is very limited which reduces their usability.

The researchers in [12] generated test cases from the use case specifications of the system. They combined Natural Language Processing (NLP) and constraint solving to extract behavioral information from use case specifications. They have achieved promising results but it was not clear how to tackle the non-functional requirements of the system.

Non of the previously mentioned approaches has specifically addressed the safety requirements of the system nor was designed to handle the specifications of a self-adaptive systems, which have to describe the adaptation strategies of the system against the uncertainty of the domain. With our proposed approach, we will find a feasible method to auto-generate test cases from SAS system specifications to test and verify the safety requirements as we maintain traceability to the safety analysis entities.

To summarize, we form the main questions we answer as the following:

1) How to mitigate safety requirements in adaptation strategies representations of SAS?
2) How to assure that the safety requirements of SAS are consistent and traceable through the design time?
3) How to auto-generate test cases that can validate the safety requirements of SAS at design time with the help of a simulator?

 ISBN: 978-1-61208-781-8

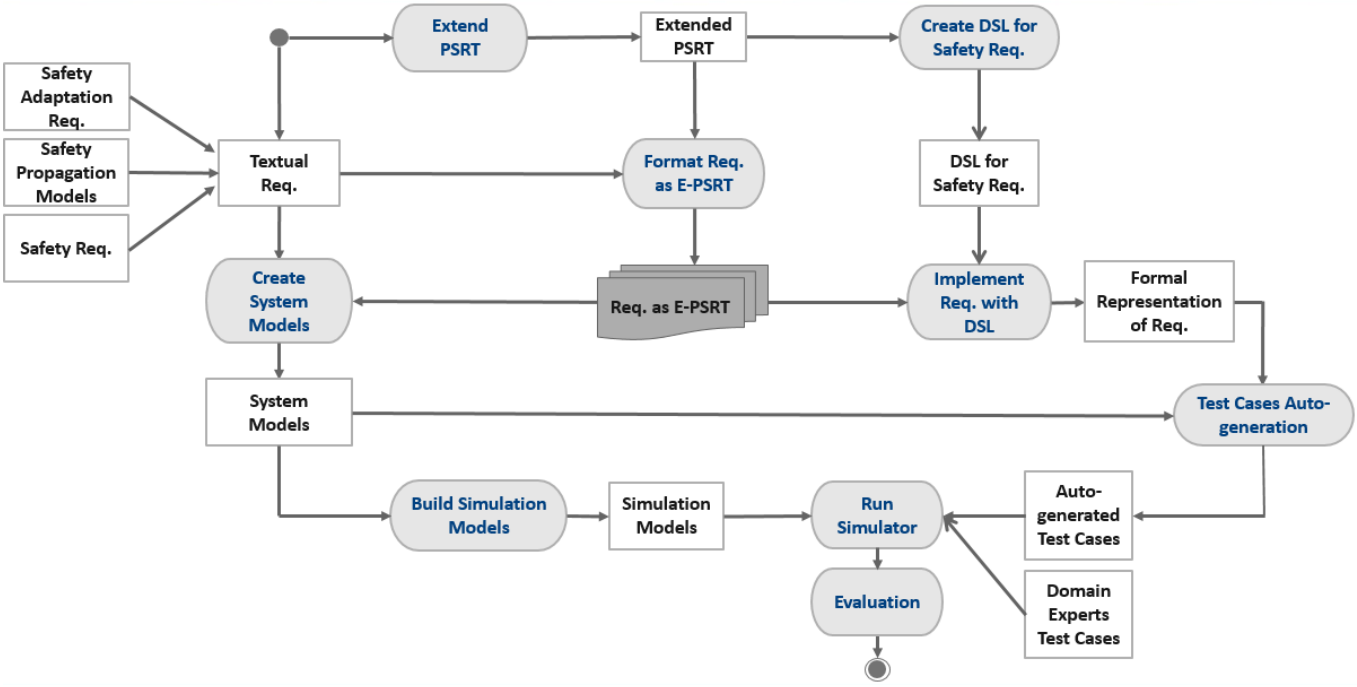

Figure 1. Proposed Approach for Validating Safety Requirements of SAS at Design Time

III. Proposed Solution

Our proposed solution to the problems described in Section II, is a comprehensive approach which consists of nine steps. Each step depends on the intermediate result from a predecessor step as illustrated in Figure 1. The main goal of this approach is to provide a means to evaluate the safety requirements of a self-adaptive system at design time and to assess the system behavior in critical events. That also gives a clear guidance about the quality of the architecture design and its adherence to the overall safety requirements of SAS. In the following, a clearer description of each of the steps is provided:

- Extend PSRTs: originally, PSRTs explained in [4], are templates which provide guidelines on how to specify the safety requirements of a system. The proper usage of PSRT templates assures consistency and establishes traceability between the safety requirements, the failure propagation models, and the system architecture. However, PSRTs weren't designed with self adaptation in mind. That is why, we start by extending the PSRTs to Extended PSRTs (E-PSRTs). E-PSRT would provide a thorough template to help specifying the safety requirements of the different adaptation scenarios and strategies of SAS systems.
- Format Requirements as E-PSRTs: in this step, we format the safety requirements of the self-adaptive system as E-PSRTs. This enable us, at design time, to identify the inconsistent safety requirements and then to update the requirements document, failure propagation models, and the system's architecture, accordingly.
- Create a Domain Specific Language (DSL): DSL is usually created to solve specific problems in a particular domain. In this step, we aim at designing a language which facilitates the representation of adaptation scenarios as well as the rules and accepted patterns of the safety requirements of a self-adaptive system. That will enable us to precisely parse the safety requirements from the E-PSRT templates to a formal structure. We can inject failures to specified safety guards requirements.
- Implement Requirements with a DSL: In this step, we implement the requirements that we formatted as E-PSRTs using the DSL that we have created. The output of this step is a formal representation of the adaptation strategies of SAS. Later on, we use the resulted representation to auto-generate test cases for testing the system along with test cases proposed by domain experts.
- Create System Models: in this step, we create the system models and architecture starting from the system requirements. This step can be conducted manually or in a semi-manual fashion. It is out of the focus of our approach to provide an automation of this step.
- Build Simulation Models: in this step, we build the simulation models that we can later run by a simulator. We build the simulation models based on the system models we created form the previous step. This step is conducted manually.
- Auto-generate Test Cases: in this step, we use the formal representation of the safety requirements and adaptation strategies of SAS and the manually created system models to auto-generate adequate test cases. The automation of the test case generation process provides a means

 ISBN: 978-1-61208-781-8

to ensure that the test cases have been derived in a consistent and objective manner and that all system's safety requirements have been covered.
- Run Simulator: in this step, we run the simulation models of the system in a simulator. Then, we feed the simulator with the auto generated test cases along with the domain experts' test cases and monitor the system's behavior.
- Evaluation: finally, we compare the expected behavior of the system with the resulted/simulated behavior. The found deviations, if any, raise an alarm that the system could encounter unpredicted and hazardous behaviors. We can trace the behaviors back to the initial requirements and perform the needed updates.

By performing the previous steps, we can validate the safety requirements of a SAS at design time. When locating design flows, they could be traced back to either the requirements of the system or to its safety analysis.

IV. Research Validation

We need a proper use case to validate our proposed approach. In the world of self-adaptive and smart systems, there exist a lot of exemplary benchmarks especially for testing object detection and object tracking algorithms. This type of benchmarks are designed to test the end behavior of an implemented system, since we are focusing on design time validation of the system. They are not appropriate to validate our approach. We need a comprehensive use case that fully describes the requirements and the system architecture.

A. Use Case

Safe Adaptive Software for Fully Electric Vehicles (SafeAdapt) is a project conducted by Fraunhofer IKS [13]. The main concept behind SafeAdapt is to develop a novel Electric/Electronic architecture based on adaptation to achieve safety, reliability, and cost efficiency in future Fully Electric Vehicles (FEVs). SafeAdapt has the goal of building adaptable systems in safety-critical environments that comprises methods, tools, and building blocks for safe adaptation.

Side by side through SafeAdapt project, a detailed use case has been built to verify the architecture of the end system. This use case is fully described through several deliverables of the project and it evolves as the project progresses. That makes it a perfect fit to evaluate our approach since it clearly covers the first two phases of the system development, namely requirements elicitation and system design.

B. Approach Validation

To properly validate our approach, we need to run the simulator twice. The first run would assess the system behavior and its hazardous encounters without applying the approach. The inputs of the simulator in this run are the simulation models and the domain experts' test cases. The second simulator run would have the system models and the auto generated test case along with the domain experts' test cases. In both runs, we will assess the design flows of the system and the unpredicted hazards that the system could encounter. The comparison between the simulator runs would provide a good gaudiness of the applicability of the approach and the improvements that it adds to the validation of safety requirements of SAS at design time.

V. Conclusion

Building adaptable systems in safety-critical environments is a challenging task. Our proposed approach addresses some of these challenges in requirements elicitation and system design phases. We first tackle the problem of specifying safety requirements of SAS and how we reflect them in the adaptation strategies while keeping them traceable to the safety analysis models and architecture components. Then we generate adequate test cases to test the expected behavior of the system. To get an early feedback before starting with system implementation, we run the generated test cases at design time with the help of a powerful simulator.

References

[1] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel, "Relax: a language to address uncertainty in self-adaptive systems requirement," *Requirements Engineering*, vol. 15, no. 2, pp. 177–196, 2010.

[2] Y. Zhuoqun, Z. Li, Z. Jin, and Y. Chen, "A systematic literature review of requirements modeling and analysis for self-adaptive systems," in *Requirements Engineering: Foundation for Software Quality*. Springer International Publishing, 2014, pp. 55–71.

[3] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "The requirements problem for adaptive systems," *ACM Trans. Manage. Inf. Syst.*, vol. 5, no. 3, 2014.

[4] P. O. Antonino, M. Trapp, P. Barbosa, and L. Sousa, "The parameterized safety requirements templates," in *2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability*. IEEE, 2015, pp. 29–35.

[5] D. Schneider and M. Trapp, "Conditional safety certification of open adaptive systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 8, no. 2, 2013.

[6] H. J. Goldsby *et al.*, "Goal-based modeling of dynamically adaptive system requirements," in *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*, 2008, pp. 36–45.

[7] S. J. Cunning and J. W. Rozenblit, "Automatic test case generation from requirements specifications for real-time embedded systems," in *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, vol. 5, 1999, pp. 784–789 vol.5.

[8] ISO, "Road vehicles – Functional safety," 2011.

[9] T. Yue, S. Ali, and L. Briand, "Automated transition from use cases to uml state machines to support state-based testing," in *Modelling Foundations and Applications*. Springer Berlin Heidelberg, 2011, pp. 115–131.

[10] T. Yue, L. C. Briand, and Y. Labiche, "An automated approach to transform use cases into activity diagrams," in *Modelling Foundations and Applications*. Springer Berlin Heidelberg, 2010, pp. 337–353.

[11] G. Carvalho *et al.*, "Nat2testscr: Test case generation from natural language requirements based on scr specifications," *Science of Computer Programming*, vol. 95, pp. 275 – 297, 2014.

[12] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, "Automatic generation of system test cases from use case specifications," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ser. ISSTA 2015. Association for Computing Machinery, 2015, p. 385–396.

[13] F. IKS. (2019) Project SafeAdapt safe adaptive software for fully electric vehicles. [Online]. Available: http://www.safeadapt.eu

 ISBN: 978-1-61208-781-8