Supporting Early-Safety Analysis for Safety-Critical IoT Systems Exploiting Model-Driven Engineering

Diego Clerissi[2], Juri Di Rocco[1], Davide Di Ruscio[1], Claudio Di Sipio[1], Felicien Ihirwe[1], Leonardo Mariani[2], Daniela Micucci[2], Maria Teresa Rossi[2] and Riccardo Rubei[1]

[1]University of L'Aquila, Italy
[2]University of Milano - Bicocca, Italy

1. Introduction and Aim

IoT systems are subject to different kinds of failures, either being internally generated or caused by the surrounding environment. Such failures may affect not only the correctness of the system but also the environment in which they operate. Consider, for instance, Smart Irrigation Systems: they monitor parameters related to weather and soil to automatically irrigate crop fields based on the data collected. A failure affecting the behavior of those IoT systems may cause a waste of water or loss to the farm's production. Since IoT systems are composed of components of different natures (e.g., temperature/humidity sensors, cloud servers, and irrigation systems), studying how a failure (e.g., caused by a malfunctioning component) may propagate within a system and impact its behavior can be extremely challenging, further than being of high importance.

2. Model-Based Test-Driven Safety Analysis

Fault-Tree Analysis (FTA) [1] allows us to analyze how failures propagate within a system to anticipate possible misbehaviours. To be effective, FTA requires accurate knowledge about how failure may propagate across the individual components. To address this limitation, we propose *Model-Based Test-Driven Safety Analysis*: an empirical approach that semi-automatically allows discovering how failures propagate at the level of the individual components exploiting testing.

In a nutshell, the proposed approach consists of three main phases (Figure 1). The *IoT System Modelling* concerns a tool-supported specification of a system model, annotated with information about how failures can be originated and propagated by components. The *IoT System Testing* phase exploits the information in the model to execute the individual components, while injecting failures on input ports, and checking how they propagate to output ports. This phase can both confirm/disprove existing failure propagation rules (soundness check) and discover new ones (completeness check). The *FT Generation and Analysis* studies how failures propagate and impact the system, according to the available rules. The outcome of the analysis can be used to refine the system, and its model, to finally achieve a more reliable IoT system. The evolution of the system and the model shall trigger additional IoT System testing that may, in turn, discover new rules, which requires re-running FTA until the resulting behavior of the system is satisfactory.

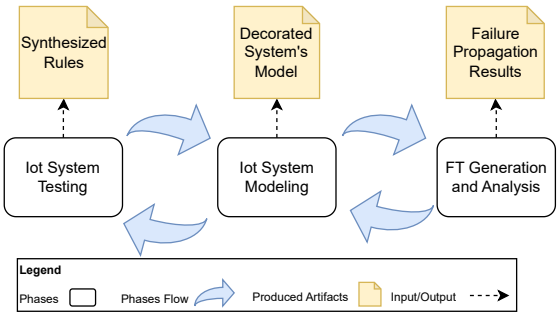

Figure 1: Model-Based Test-Driven Safety Analysis.

2.1. IoT System Modeling

The modeling and analysis approach runs on top of CHESSIoT [2], a model-driven environment to support the design and analysis of IoT systems. CHESSIoT provides a UML/SysML profile extension to reflect the constructs and semantics present in the IoT system-level architectures. The system-level model includes all of the system's major functional components and interconnections. Ports enable interactions among components and are fundamental for determining error propagation paths.

Once the model is completed, the safety engineer de-

diego.clerissi@unimib.it (D. Clerissi); juri.dirocco@univaq.it (J. Di Rocco); davide.diruscio@univaq.it (D. Di Ruscio); claudio.disipio@univaq.it (C. Di Sipio); jeanfelicien.ihirwe@graduate.univaq.it (F. Ihirwe); leonardo.mariani@unimib.it (L. Mariani); daniela.micucci@unimib.it (D. Micucci); maria.rossi@unimib.it (M. T. Rossi); riccardo.rubei@gmail.com (R. Rubei)
0000-0002-9421-8566 (D. Clerissi); 0000-0001-7116-9338 (J. Di Rocco); 0000-0002-5077-6793 (D. Di Ruscio); 0000-0002-0877-7063 (C. Di Sipio); 0000-0002-4463-6268 (F. Ihirwe); 0000-0002-9421-8566 (L. Mariani); 0000-0002-9421-8566 (D. Micucci); 0000-0003-0273-7324 (M. T. Rossi); 0000-0002-9421-8566 (R. Rubei)

rives and annotates the failure behavior rules for each component following the Failure Propagation Transformation Calculus (FPTC) [3] notation. The Failure Logic Analysis (FLA) [4] considers the component functional decomposition down to fundamental sub-functions, which are then systematically evaluated to identify all their potential failure modes. The analyst must establish error propagation or transformation rules for each possible input failure or internal failure of the component.

Based on its nature, a component can propagate a failure (carrying a failure from input to output), transform a failure (changing the nature of a failure from input to output), act as a source of failure (creating a failure despite no failure in input), or act as a sink (avoiding the failure to be either propagated or transformed). The following three abstract categories of failure modes are assessed: *service provision failures*, such as the omission or commission of the output; *timing failures*, such as the early or late delivery of the output; and *value domain failures*, such as the output value being out of a valid range, stuck, or exhibiting erratic behavior. In addition, a *noFailure* annotation is used to indicate a no-failure mode at the input port. At this point, the FLA transformation can be launched, generating the FLA model and later being used for Fault-Tree generation and analysis.

2.2. Fault-Tree Generation and Analysis

The Fault-Tree generation is performed through a series of model-to-model transformations from the FLA model to a series of Fault-Tree (FT) models. An FT is generated for each of the failures that propagate to the targeted output port of the system and contains logical networks of events and corresponding gates that together form a failure representation tree reflecting the system's failure behavior set by the user as well as the system's functional architecture. Each FT event has its unique identities in the tree and can be of type basic, intermediate, external, or undeveloped, depending on the stage at which it had manifested. From the FT, it is possible to graphically trace down the influence of individual failures on the components or system-level hazardous failures.

As the system gets bigger and more complex, which in turn requires a large number of rules to better cover all possible failure scenarios, it is inevitable that the generated FTs become even bigger and harder to grasp. To tackle such a challenge, CHESSIoT offers a *"Qualitative"* fault tree analysis that aims at analyzing the generated FTs by keeping only the essential representations. This is done by removing unnecessary paths as well as redundancies. In addition, CHESSIoT offers *"Quantitative"* analysis that automatically calculates the failure probabilities of an entire system from its constituent parts' failure event probabilities by means of logic probabilities calculation mechanism techniques [5].

2.3. IoT System Testing

IoT System Testing systematically tests in isolation the IoT components present in a system in order to (i) *validate* existing failure propagation rules (e.g., present in the model) and (ii) *discover* new failure propagation rules. Components under test are isolated by adding stubs that simulate the components communicating with them. The stubs are used to generate inputs in a controlled way, whereas monitoring probes are introduced to capture the output produced by the components under test.

In order to discover failure propagation rules, the testing phase exploits differential testing, that is, it records a base execution, collected in a normal scenario, and then produces many mutated executions, each one differing from the base execution for a failure injected on an input port. The comparison of the output produced by the base execution and a mutated execution reveals if and how the input failure propagated to the output. Systematically injecting different types of failures on the input ports, for different base executions and for different components, is thus exploited to systematically discover failure propagation rules. More rigorously, every component C is tested against $FT = N^I$ failures, where N is the number of supported failure types and I is the number of input ports where these failures can be injected. Moreover, to consider the case failure propagation may depend on the state of the component, the FT failures are executed against S possible states. Our implementation currently supports the injection and detection of *timing failures* and *value domain failures*. The set of states S to be used for failure discovery can be defined by the user or can be automatically extracted from an existing test suite by selecting those states that produce observable differences in the output generated by a component. Our tool implementation currently supports Proteus components[1].

3. Ongoing and Future Work

We are currently investigating empirically three key research questions about the effectiveness of our approach.

- **RQ1** Are rules specified by domain experts correct w.r.t. the real failure propagation patterns?
- **RQ2** Are rules specified by domain experts completely specifying the behavior of IoT components?
- **RQ3** Can our approach compensate for inaccuracies in the rules provided by domain experts?

We are also working to increase the automation level of the approach and support additional environments and languages.

[1] https://www.labcenter.com/

References

[1] L. Xing, S. V. Amari, Fault tree analysis, in: K. B. Misra (Ed.), Handbook of Performability Engineering, 2008, pp. 595–620. doi:10.1007/978-1-84800-131-2_38.

[2] F. Ihirwe, S. Mazzini, P. Pierini, A. Debiasi, S. Tonetta, Model-based analysis support for dependable complex systems in CHESS, CoRR abs/2009.06089 (2020).

[3] M. Wallace, Modular architectural representation and analysis of fault propagation and transformation, Electronic Notes in Theoretical Computer Science 141 (2005) 53–71. doi:https://doi.org/10.1016/j.entcs.2005.02.051.

[4] B. Gallina, M. A. Javed, F. U. Muram, S. Punnekkat, A model-driven dependability analysis method for component-based architectures, in: Proceedings of the Euromicro Conference on Software Engineering and Advanced Applications, 2012. doi:10.1109/SEAA.2012.35.

[5] R. Ferdous, F. Khan, B. Veitch, P. R. Amyotte, Methodology for computer aided fuzzy fault tree analysis, Process Safety and Environmental Protection 87 (2009) 217–226. doi:https://doi.org/10.1016/j.psep.2009.04.004.