Exploiting Behavior Models for Availability Analysis of Interactive Systems

Maximilian Junker
Technische Universität München

***Abstract*—We propose an approach for availability analysis that directly utilizes behavior models as they occur in model-based development. The main benefits of our approach are reduced effort as no dedicated availability models need to be created as well as precise results due to the inclusion of behavior interactions.**

I. Introduction

For a wide range of software-intensive systems, the importance of availability is beyond doubt. In this paper, we focus on the availability of interactive systems. Interactive systems continously process input signals and react by producing output signals. As these systems are more and more frequently developed model-based, using tools such as Mathworks Simulink, detailed system models are increasingly available. System models have already been used for availability analysis (e.g. [1], [2]) as this relieves the availability analyst from the burden of creating additional analysis models. However, current approaches only exploit the modelled architecture or use high level abstractions of the behavior such as fault propagation. They thus are unable to capture subtle interactions between components. These interactions, however, influence the system's behavior in case of faults and hence influence the system's availability. Interviews with 15 availability experts from industry, which we conducted prior to this work, indicated that capturing dependencies between components is one of the main challenges for availability analysis.

We propose a novel approach that takes behavior models as basis for availability analysis and thus automatically also covers effects due to behavioral interaction. The core idea is to compare the behavior of a system model including faults and fault-tolerance mechanisms against a model of nominal behavior and by this comparison "measure" the availability of the system. We have created initial tool support of our approach and did a proof-of-concept availability analysis using a model of an industrial train control system [3].

II. The System Model

Our notion of a system is the following: A systems possesses a (syntactic) interface represented as input and output channels which may transmit messages. A system exhibits a behavior that can be observed at its interface in terms of incoming and outgoing messages. Mathematically we capture such systems as functions mapping streams of input messages I to streams of output messages O, as it has been defined in [4] and probabilistically extended in [5]:

$$S : I^{\infty} \rightarrow O^{\infty}$$

This work is partially supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS12057.

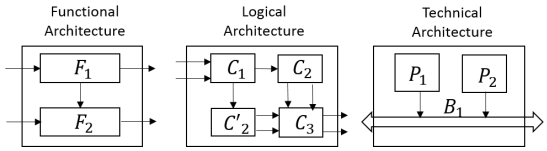

Fig. 1. The functional architecture describes system functions F_i. The logical architecture describes the decomposition into components C_i. The technical architecture describes the hardware platform, e.g. processors P_i and a bus B_1.

Behavior may be described in different ways, for example using (probabilistic) I/O automata.

We describe systems from the following three different viewpoints which results in three different types of architectures. See Fig. 1 for a schematic overview.

The *functional architecture* is a complete description of the nominal black-box behavior of a system. The functional architecture is very close to the original system requirements. The structuring follows only functional criteria and not considerations such as reuse, maintainability or performance. The functional architecture may omit technical details, such as fault-tolerance mechanisms. It may thus be considered an *executable* functional specification.

The *logical architecture* also describes a system's behavior. The system is decomposed into components along mostly non-functional considerations. Details such as fault-tolerance mechanisms are included into the logical architecture. Together, the components of the logical architecture should show a behavior that corresponds largely to the behavior of the functions described in the functional architecture.

The *technical architecture* is a description of the technical platform that the system is deployed on. Thus, elements in this architecture are processing units, busses, sensors, actuators, etc. The technical architecture may include information on the fault characteristics of technical devices that influences the behavior of the components that are deployed on these devices.

Using models of the logical and technical architecture we can obtain an *executable* model of the system's actual behavior that encompasses behavior in the presence of faults and fault-tolerance mechanisms and furthermore includes a large part of the possible component interactions.

III. Availabiltiy Analysis

Overview The core idea of our approach is to determine the availability of a system by comparing its behavior with its specification and derive an availability metric from the extent of the deviations observed over time. In our case the

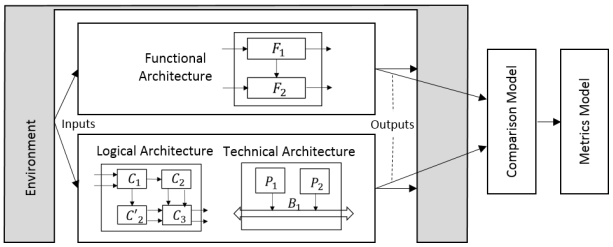

Fig. 2. Overview over the models used for availability analysis.

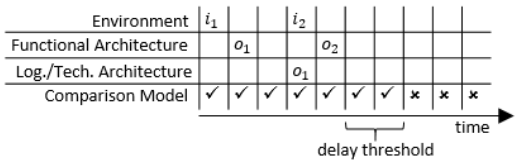

Fig. 3. Example execution of the different models with a comparison model for the failure mode "message delay".

system is modeled in the logical and technical architecture, the specification is represented through the functional architecture. As availability may depend on the context, we perform the comparison within an environment model that simulates external systems or users. As the comparison may be complex, for example to accommodate different failure modes, we have an explicit model describing the comparison rules. Finally, a metrics model defines the specific availability metrics that are calculated for the analysis. Fig. 2 provides an overview over the models used in our approach.

Environment Model The environment model describes assumptions on the behavior of the environment. This is comparable to an operational profile that describes the probability of certain inputs. Additionally the environment model describes how the environment reacts to the system outputs.

System Models Both types of system models, the functional architecture and the logical/technical architecture are configured "in parallel". That means both receive the environment's input and calculate outputs. The outputs of the functional architecture represent the desired outputs; the outputs of the logical/technical architecture represent the observed outputs.

Comparison Model The comparison model encodes the decision when an observed behavior deviates in such a way from the specified behavior that the system is considered as not operational. Comparison models are system specific, although they can be built using generic components. Consider for example the functionality responsible for displaying a system's status. Failure modes that are relevant for this functionality are for example "No status value displayed", "old status value displayed" or "wrong status value displayed". The comparison model in this case will distinguish these failure modes judging from the output signals of the functional architecture model and the logical/technical architecture model.

Metrics Model There are a range of standard availability metrics such as uptime, mean-time-between-failure, etc. In different contexts different availability metrics may be of interest. Sometimes, project-specific or domain-specific metrics are needed, especially if the analysis focuses on service-availability. The metrics model describes the metrics that should be calculated, based on the output of the comparison model.

Consider Fig. 3 for an illustrative example. The environment produces inputs i_1 and i_2. The specified reaction is o_1 and o_2. The tolerable delay are two time-units. The comparison model thus declares a failure (symbol ✗) when an output message of the logical/technical architecture model arrives with a larger delay. A simple metric model could then specify the uptime until time t as

$$U(t) = \frac{\#_t(\checkmark)}{\#_t(\checkmark) + \#_t(\text{✗})} ,$$

where $\#_t(\cdot)$ counts the occurrences of a symbol during $[0; t]$.

IV. First Experiments

Implementation In first experiments with this approach we use AutoFocus3[1] (AF3) as modeling tool. AF3 supports behavior modeling using various types of I/O automata. It also allows the creation of functional, logical and technical architectures. There is no explicit support for the comparison and metrics model, therefore we use standard modeling facilities. From the AF3 model we generate code for the prism model-checker[2]. The system models, the environment model and the comparison model can be easily represented in the prism input language as a markov decision process (i.e. a state-based model with local non-determinism and probabilistic state-transitions). We encode our metrics model in prism as reward structures together with a suitable property specification.

Case Study To evaluate the feasibility of our approach, we performed first experiments in the context of a cooperation with Siemens. There, we modeled the part of an automatic train control responsible for controlling train doors [3]. For the study we considered two system functions (human-machine-interface and propulsion release) and the complete logical/technical architecture. The architecture consists of 29 interacting components deployed on four devices. It was extended with basic fault models. Additionally an environment model was present. We further extended the original model with comparison models and metrics models denoting uptime. From this we were able to compute the uptime using prism. Our first experiments hence indicate that the approach is feasible.

References

[1] R. H. Reussner, H. W. Schmidt, and I. H. Poernomo, "Reliability prediction for component-based software architectures," *Journal of Systems and Software*, vol. 66, no. 3, pp. 241 – 252, 2003.

[2] S. Bernardi, F. Flammini, S. Marrone, J. Merseguer, C. Papa, and V. Vittorini, "Model-driven availability evaluation of railway control systems," in *Computer Safety, Reliability, and Security*, F. Flammini, S. Bologna, and V. Vittorini, Eds. Springer, pp. 15–28.

[3] W. Böhm, M. Junker, A. Vogelsang, S. Teufl, R. Pinger, and K. Rahn, "A formal systems engineering approach in practice: An experience report," in *Proceedings of SER&IPS'14*. ACM, 2014.

[4] M. Broy, "A logical basis for component-oriented software and systems engineering," *The Computer Journal*, vol. 53, no. 10, 2010.

[5] P. Neubeck, "A probabilitistic theory of interactive systems," Dissertation, Technische Universität München, München, 2012.

[1] http://af3.fortiss.org

[2] http://www.prismmodelchecker.org