ARTICLE IN PRESS SCICO:2155

Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Formal analysis of feature degradation in fault-tolerant automotive systems

Klaus Becker [*], Sebastian Voss, Bernhard Schätz

fortiss GmbH, An-Institut Technische Universität München, Guerickestr. 25, 80805 München, Germany

ARTICLE INFO

Article history:
Received 16 May 2016
Received in revised form 7 October 2017
Accepted 11 October 2017
Available online xxxx

Keywords:
Graceful degradation
Fault tolerance
Redundancy
Fail-operational
Mixed criticality

ABSTRACT

Safety critical fault-tolerant embedded systems have to react properly on failures of internal system elements to avoid failure propagation and finally a harmful external failure at the system boundary. Beside failure detection, actions for failure handling are essential to cover safety requirements. Actions reach from enabling fail-silent, fail-safe or fail-operational behavior of system elements, or also hybrids of this in a mixed criticality system design. Graceful degradation can be applied when system resources become insufficient, reducing the set of provided functional features. In this paper, we address mixed criticality and mixed reliability automotive systems. We consider mixed reliability by functional features having different fail-operational requirements. Beside pure fail-operational features, we also consider degradations of functional features, called fail-degraded features. We describe a formal system model that contains, i.a., the functional features of a vehicle, possible feature degradations, software components that realize the features, as well as the deployment of software components to execution units. We provide a structural analysis of the level of degradation on system level and feature level, which is required in scenarios of failing execution units and/or software components. Combined with this analysis, we synthesize valid deployments of software components to execution units, incorporating an adequate level of redundancy to meet the fail-operational requirements, if feasible. We apply our approach to a constructed automotive example.

1. Introduction and motivation

Many embedded systems operate in safety-critical environments, in which faults could cause errors and failures of system elements, and finally a harmful failure of the entire system. This requires that these systems can detect failing elements, such as hardware or software components, and react properly. If a functional feature is affected by a failing element, such that the feature would violate its specified behavior, one option is to shut down this feature in a fail-safe manner. However, going into a fail-safe state causes the loss of that feature. This is not acceptable for features that require fail-operational behavior. Hence, mechanisms are required to keep fail-operational features available, even if they are affected by a failed system element.

If system resources get lost due to hardware failures, the remaining resources should be used efficiently to keep those features alive that comprise the highest demand with respect to safety, reliability and availability. We use these three notions as defined in [3]. For instance, if an execution unit has to be isolated from the remaining system due to a hardware failure,

[*] Corresponding author.
E-mail addresses: becker@fortiss.org (K. Becker), voss@fortiss.org (S. Voss).

https://doi.org/10.1016/j.scico.2017.10.007
0167-6423/© 2017 Elsevier B.V. All rights reserved.

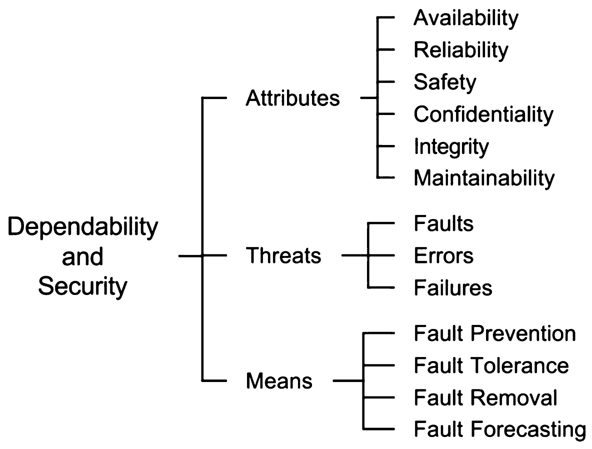

Fig. 1. Dependability and its tree of attributes, threats and means [3].

another execution unit has to be able to provide some of those features that were provided by the failing unit. However, as the remaining system-resources may become insufficient to provide the full set of features, it may be needed to explicitly deactivate or degrade some features to handle this resource limitation. This results in graceful degradation of the system.

In case a software component (SWC) permanently fails to provide its specified functionality, it has to be isolated from the residual system to avoid failure propagation and harm. We assume an underlying platform runtime environment (RTE) which is able to detect such failures, to isolate failing components, and to trigger recovery mechanisms. Without activation of a backup component, the functional features that are realized by a failing SWC cannot be provided anymore. However, redundant backups of the same SWC would not be very helpful in this scenario, as a systematic fault (e.g., a bug) would be contained in all backups. Hence, there is no use to deploy the same buggy SWC multiple times redundantly. Instead, *diversity* by alternative implementations is needed.

This paper is based on the following previous publications. In [6], we introduced a first formal model and formal constraints to calculate valid redundant deployments of software components (SWCs) to the execution units of a particular fault-tolerant system platform. In scenarios of assumed hardware failures of execution units, we formally analyzed possible actions to keep fail-operational features available. In [7], we extended the model and the analysis by communication channels between the SWCs, again considering failing execution units. In [8], we further extended the approach to support the analysis of degradations on functional feature level, as well as degradations of SWCs combined with *diversity*. Supplementary, we considered permanent failures of SWCs (e.g., due to software bugs) in the set of analyzed failure scenarios. In this paper, we extend feature degradations to support multiple subsequent degradations and integrate the model with the previous work of redundant deployments and communication channels between SWCs. We furthermore present in more detail, how we formally analyze the degradation scenarios, how we express formal constraints based on the introduced model, and how we conduct optimized design decisions as part of our analysis.

We introduce how diversity can be used by implementing two similar features, which however are not providing exactly the same specification, but the second feature is a degraded version of the first full-fledged feature. This means, a failing full-fledged feature can be substituted by another degraded feature that fulfills a subset of the original requirements, with potentially less quality of service. An example is a full-fledged functional feature that provides a steer-by-wire application of a vehicle including active assistance functions, like lane-keeping or collision-avoidance. The degraded corresponding feature may support only rudimentary manual steering, without any assistance functions.

The existence of a degraded feature is helpful, if the system resources become insufficient to provide the initial full-fledged feature. In this case, the degraded feature can be activated, assuming that this requires less resources than the corresponding full-fledged feature.

Our approach comprises a formal system model and a set of formal constraints describing the validity of deployments and degradation scenarios with respect to a given safety-concept. The model and the constraints characterize an arithmetic logical problem that can be solved, for instance, by *satisfiability modulo theories* (SMT) solvers, like Z3 [17,11].

1.1. Fundamentals about dependable, gracefully degrading systems

The *dependability* of a system is defined as the ability to avoid service failures that are more frequent and more severe than acceptable [3]. Dependability is an integrated concept that encompasses dependability *attributes*, *threats* and *means*, as shown in Fig. 1.

The topic of this paper is related to the threats of errors and failures; the attributes of availability, reliability and safety; as well as the means of fault tolerance.

Fault-tolerance is the ability of a controlled system to maintain control objectives, despite the occurrence of a fault [12]. According to [54], a software system is fault-tolerant if it is able to respond gracefully to failures at run time.