From Fault Injection to Mutant Injection: The Next Step for Safety Analysis?

Guillermo Rodriguez-Navas[1], Patrick Graydon[1], and Iain Bate[1,2]

[1] Dept. of Engineering, Design and Technology. Mälardalen University, Sweden
guillermo.rodriguez-navas@mdh.se, patrick.graydon@mdh.se
[2] Dept. of Computer Science. University of York, UK
iain.bate@cs.york.ac.uk

Abstract. Mutation testing has been used to assess test suite coverage, and researchers have proposed adapting the idea for other uses. Safety kernels allow the use of untrusted software components in safety-critical applications: a trusted software safety kernel detects undesired behavior and takes remedial action. We propose to use specification mutation, model checking, and model-based testing to verify safety kernels for component-based, safety-critical computer systems.

Keywords: Safety-critical systems, safety analysis, mutation testing, component based design.

1 Introduction

Mutation testing has been used to assess test suites [1]. Test software applies *mutation operators* to the software, creating mutant versions with known forms of implementation faults. The more mutants a given test suite detects, the more confidence we can have in the tested software. Researchers have applied mutation to specifications, interfaces, and contracts to assess coverage of faults introduced in the specification and design phases. Safety researchers have even suggested mutations based on Hazard and Operability studies (HAZOP) [2].

Software components are frequently used out of context. However, it is not possible to verify a component for adequately safe use in all possible contexts and applications [3]. Thus, safety-critical, component-based software systems must tolerate unexpected behavior from components re-used out of context.

Safety kernels permit using untrusted software components—such as COTS or SOUP—in safety-critical applications [4]. The trusted safety kernel wraps the untrusted component, detects undesired behavior, and takes remedial action as appropriate. For example, a safety kernel might detect a postcondition violation, restart the offending component, and flag its output as potentially erroneous.

Model checking allows the exploration of whether certain key properties of the system hold, e.g. those enforced by the safety kernel. When combined with mutation testing, we can ascertain whether the key safety properties hold in the presence of failures which is important when assuring the safety and dependability of critical systems. Despite the model being used being an abstract form of the final system, it allows the subsequent development steps to be de-risked and provide invaluable evidence as to the ability of the safety kernel to prevent hazards. Finally, model-based testing will validate the implementation.

2 Our Approach

Verifying the safety kernel specification. We assume: (a) a specification of each component and its functional and temporal behavior, e.g. in EAST-ADL with suitable extensions; (b) a specification of the safety kernel's behavior; and (c) a specification at the system level of the hazardous conditions to avoid. Our challenge is to verify that *if* the safety kernel behaves as specified, no plausible single failure of a wrapped component will put the system into a hazardous state.

Validating the safety kernel implementation. It is possible that the safety kernel will be faulty. This is true even if it is automatically generated from a correct specification: compilers and other tools might be buggy. Our challenge here is to automatically generate test cases to validate the implemented safety kernel. We have identified three test mechanisms: (1) mutating the wrapped components' code (if available); (2) modifying values passing through framework communication channels (where applicable); and (3) generating stub components.

Research challenges. First challenge is extending the nominal behavior of the components with both a set of plausible failures and their corresponding repair mechanisms. Second challenge is to automatically link these new potential behaviors with the specified safety kernel. Third challenge is introducing some kind of behavioral propagation of failures into the models, not based on transformation rules but on the real evolution of the components. Our ambition is to be able to introduce all these features directly into the timed automata models with as little user intervention as possible. The challenges associated to the validation of the implementation will be studied during a second phase of this work.

Acknowledgment. This work was partially supported by the Swedish Foundation for Strategic Research (SSF), under grant number RIT10-0070 (SYNOPSIS).

References

1. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering 37(5), 649–678 (2011)
2. Araujo, R., Maldonado, J., Delamaro, M., Vincenzi, A., Delebecque, F.: Devising mutant operators for dynamic systems models by applying the HAZOP study. In: Proc. of the 6th Int'l Conference on Software Engineering Advances (2011)
3. Rushby, J.: Modular certification. Technical Report CR-2002-212130, National Aeronautics and Space Administration, Hampton, VA, USA (December 2002)
4. Wika, K.G., Knight, J.C.: On the enforcement of software safety policies. In: Systems Integrity, Software Safety and Process Security: Proceedings of the 10th Annual Conference on Computer Assurance (COMPASS), pp. 83–93 (June 1995)