Safety of Component-Based Systems: Analysis and Improvement using Fujaba4Eclipse [*]

Matthias Tichy, Stefan Henkler, Matthias Meyer, and Markus von Detten
Software Engineering Group, Department of Computer Science,
University of Paderborn, Paderborn, Germany
[mtt|shenkler|mm|mvdetten]@uni-paderborn.de

ABSTRACT

Todays embedded and safety-critical systems incorporate increasing amounts of software. Consequently, the software architecture and its connection to hardware elements have a big impact on the safety of those systems. We present in this paper an approach and its implementation in the Fujaba4Eclipse environment for the analysis and improvement of component-based systems w.r.t. their safety which specifically exploits the software and system structure.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Reliability and life testing; D.2.11 [**Software Engineering**]: Software Architectures—*languages, patterns*

General Terms

Design, Measurement

Keywords

Safety, Fault Tolerance, Fujaba, Structural Transformations, Failure Propagation, Hazard Analysis

1. INTRODUCTION

Software has become the driving force in the evolution of many technical systems and in some areas grows at an exponential rate. As a consequence, system engineers face a dramatically increasing complexity due to the cooperation of beforehand isolated functions, as e.g. in the domain of automotive software [4]. To counter the effect of growing complexity, systems are often built in a component-based fashion. A concrete system then is a specific composition of reusable components.

Technical systems are often employed in a safety-critical context. The software components of such a system, in particular their interaction and their distribution throughout the system, have a tremendous impact on its safety. Thus, when reasoning about the safety of a system, the whole system architecture has to be taken into account.

In [1], we presented approaches and their implementation in the Fujaba research prototype for the modeling of the structure as well as real-time and hybrid behavior of safety-critical embedded systems. In both approaches, formal verification is used to detect *systematic faults* in the behavioral models.

In the remainder of this paper, we present a complementary approach [8] and its implementation in the Eclipse version of the Fujaba research prototype to tackle *random faults*. The impact of random faults by their propagation throughout the system architecture is explored using a component-based hazard analysis. The system architecture can then be improved by the application of fault tolerance techniques which are formalized using graph transformations. The next section shows an overview of the approach (cf. Figure 1). We conclude in Section 3 with an outlook on future work.

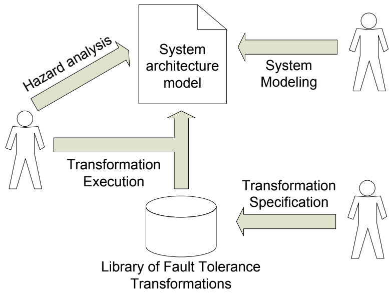
Hazard analysis
System architecture model
System Modeling
Transformation Execution
Transformation Specification
Library of Fault Tolerance Transformations

Figure 1: Approach overview

2. THE APPROACH

We use UML 2.0 components and deployment diagrams to model the system architecture. Each component type is extended by a failure propagation specification (similar to the fault pathology of Laprie [5]) using Boolean logic. In general, a failure propagation consists of a set of outgoing failure variables, a set of incoming failure variables, a set of internal error variables, and failure dependencies. The specified failures and errors are typed and we distinguish the general failure classes crash, timing, and value failure. The approach additionally supports user-defined failure classifications.

After modeling the component types and their failure propagations, the software components are deployed on hardware components. On this instance view, hazards are specified. To describe the occurrence of hazards, we use standard fault tree analysis. Hence, the hazardous event is shown as the

[*] This work was developed in the course of the Special Research Initiative 614 – Self-optimizing Concepts and Structures in Mechanical Engineering – University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

ICSE'08, May 10–18, 2008, Leipzig, Germany.
ACM 978-1-60558-079-1/08/05.

top of a fault tree which is caused by a combination of failure variables and AND/OR operators.

The system failure propagation is a combination of the failure propagations of all component instances in the deployment diagram with automatically inferred failure propagations of the connectors between components. The system failure propagation is then combined with the hazard definition for the hazard analysis.

The approach supports two types of hazard analysis: 1) which hazards result from a set of given basic errors (bottom up) as well as 2) which errors must occur in order for a given hazard to happen [3] (top down). Our approach employs binary decision diagrams (BDDs) for efficient operation.

The top-down analyis uses the BDD representation of the system failure propagation to compute the event combinations (prime implicants) which lead to the hazard. The prime implicants of a Boolean formula are of special interest in a hazard analysis since they denote smallest hazard scenarios. In addition to the possibility of a hazard occurrence, its probability is computable, if (independent) probabilities are known for the basic errors. This probability is recursively computed on the BDD as shown in [3].

We employ a simulation of the failure propagation in the bottom-up analysis. Additionally, the simulation enables the developer to visually see the propagation path of the errors through the system architecture. The simulation starts with the given set of errors and recursively evaluates and executes the failure propagations and hazard conditions.

The analysis step identifies which errors in which component instances of the modeled system ultimately lead to safety-critical situations. In order to keep the system operating as safely as possible, such situations should be avoided. The triggering errors are inherently unavoidable. Their effect on the system, however, can be minimized using fault tolerance techniques (cf. [7]).

We support the semi-automatic application of fault tolerance techniques to an existing system model. The steps which are necessary to implement such a technique in an existing model are formally specified by a transformation language which combines activity diagrams and graph transformations [2]. The transformations are specified w.r.t. the metamodel of the system model.

Like activity diagrams, transformations allow to connect activities, i.e. graph transformation rules, in complex control flow consisting of sequences, alternatives, and iterations. Unless a graph transformation rule is explicitly specified to be iterated, it is executed only once when reached by the control flow. Transformations may call other transformations to facilitate composition and reuse.

3. CONCLUSIONS AND FUTURE WORK

We presented an approach for the analysis and improvement of the safety of component-based systems. The approach is prototypically implemented as a set of plugins for the Fujaba4Eclipse environment.

The approach and the accompanying research prototype can be improved in several ways. It would benefit from checking the correctness of the modeled abstract failure propagation behavior of each component w.r.t. its functional behavior. Additionally, the failure propagation might be (semi-)automatically inferable from the functional behavior. If new component types are introduced by the application of a transformation, the real-time or hybrid behavior of those components might be automatically synthesized from the communication protocols of the involved components as discussed in [8]. The correctness of the transformations themselves is important. We are currently working on verification techniques which allow us to prove that transformations do not violate certain structural properties when applied to a model (cf. [6]).

Acknowledgments. Special thanks go to all students which realized parts of this research prototype in their master's theses and project groups. We thank Martin Hirsch for comments on earlier versions of this paper.

4. REFERENCES

[1] S. Burmester, H. Giese, S. Henkler, M. Hirsch, M. Tichy, A. Gambuzza, E. Müch, and H. Vöcking. Tool support for developing advanced mechatronic systems: Integrating the fujaba real-time tool suite with camel-view. In *Proc. of the 29th International Conference on Software Engineering (ICSE), Minneapolis, Minnesota, USA*, pages 801–804, May 2007.

[2] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story diagrams: A new graph rewrite language based on the unified modeling language. In *Proc. of the* 6^{th} *International Workshop on Theory and Application of Graph Transformation (TAGT), Paderborn, Germany*, LNCS 1764, pages 296–309. Springer Verlag, November 1998.

[3] H. Giese and M. Tichy. Component-Based Hazard Analysis: Optimal Designs, Product Lines, and Online-Reconfiguration. In *Proc. of the 25th International Conference on Computer Safety, Security and Reliability, Gdansk, Poland*, LNCS. Springer Verlag, September 2006.

[4] K. Grimm. Software technology in an automotive company: major challenges. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 498–503, Washington, DC, USA, 2003. IEEE Computer Society.

[5] J. C. Laprie, editor. *Dependability : basic concepts and terminology in English, French, German, Italian and Japanese [IFIP WG 10.4, Dependable Computing and Fault Tolerance]*, volume 5 of *Dependable computing and fault tolerant systems*. Springer Verlag, Wien, 1992.

[6] M. Meyer. Pattern-based Reengineering of Software Systems. In *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006), Benevento, Italy*, pages 305–306. IEEE Computer Society, October 2006.

[7] N. Storey. *Safety-Critical Computer Systems.* Addison-Wesley, 1996.

[8] M. Tichy. Pattern-based synthesis of fault-tolerant embedded systems. In *Proc. of the Doctoral Symposium of the Fourteenth ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE), Portland, Oregon, USA*, pages 13–18, Nov. 2006.