A Methodology for Benchmarking-based Abstract Fault Modeling

Gergely PINTÉR, András PATARICZA
Budapest University of Technology and Economics
Department of Measurement and Information Systems
{pinterg, pataric }@mit.bme.hu

Abstract

This paper presents a method and environment for embedding the high-level manifestation of low-level (physical) faults into the behavioral software model in the form of in abstract reusable fault models. The main focus is on the application of intelligent data processing methods for extracting important phenomena from the observations in fault injection experiments.

1. Introduction

Designs targeting dependable services rely mainly on the use of redundant design patterns. Although high-level redundancy assures a good fault tolerance (FT), a wide field of applications cannot tolerate a large overhead caused by a high degree of redundancy like non-mission critical embedded systems. In these cases a very faithful fault model has to be estimated and dedicated FT measures have to exactly fit to it in order to avoid the introduction of an unacceptable overhead.

Since modern software development methodologies start with high-level modeling, FT mechanisms have to be included into the models from the early phases of the development on. This approach necessitates a faithful mapping of the implementation dependent physical faults to the implementation independent behavioral level. Fault metamodels can describe fault mechanisms in a generalized form. This can be used to generate the faulty instances of the target system as mutations by distorting the fault-free behavioral model.

The core idea of our approach is based on experimental investigation of typical (benchmark) applications in presence of low level faults injected into the system. These observations are used for a generalization of the impacts of implementation steps (code generation, compilation etc.) to the behavioral level description of the faults.

The discovered behavior can be seen as a fault meta-model, a set of transformation rules that can be applied to the concrete behavioral model resulting in the concrete fault model (e.g. such fault meta-model describes control faults by including some specific additional edges into the statechart of the fault-free case, thus representing fault induced illegal state transitions).

The extraction of the fault meta-models needs the discovery of "typical distortions" in the huge log files from fault injection campaigns. Our approach did use data mining to automate this step.

2. Method overview

Our method consists of four main steps: (i) creation of a behavioral model of the benchmark application, (ii) source code-level instantiation, (iii) fault injection, and (iv) postprocessing of the measurement results.

UML statcharts were used for *behavioral modeling.* These statecharts provide the basis for (i) automatic code generation and (ii) reference information for failure detection controlling the collection of log data as well.

The abstract models were *instantiated to the source code level* by automatic code generators. Two tools were used in our approach: the i-Logix Rhapsody and a simple dedicated, fine tunable code generator developed by us.

The experiments were performed on a *simulated faulty platform* consisting of a software-implemented fault injector and a statechart-level control flow checking mechanism. The tested programs were instrumented to send signatures assigned to states of the UML statechart.

Deviations from the reference behavior specified by the statechart for the fault-free case were detected and stored in a database by a software-implemented watchdog processor [2]. The database records contained the exact circumstances of the fault injection run (original register value, inverted bit, etc.) and the result of the test run (failure mode, like a crash of the tested program or an illegal state transition).

Single bit inversion faults were injected into the Instruction Pointer (IP) during the run of the benchmark application. 200000 targeted fault injection runs were performed each hitting the execution of the statechart.

Processing of experimental results aims at the generalization of observed fault effects. The majority of faults results in straightforward control flow distortions. However a minor, but non-negligible fraction exposes complex failure phenomena, which have to be generalized to the high modeling level. Please note, that traditional statistical evaluation methods simply suppress these secondary failure mechanisms as noise. In our experiments the first preprocessing step removed the observations corresponding to trivial failure modes in order to focus on the secondary mechanisms.

Classification, a special kind of predictive modeling was used for extracting the complex phenomena from the fault injection logs by the IBM Intelligent Miner for Data [3]. The method clusters the observation results into groups composed of internally strongly similar records but exposing a high level of difference between records in different groups.

The analysis result is presented as a binary decision tree with logic formulae over the record attributes assigned to the branches. The root of the tree represents the entire set of data. Branches gradually partition the data into smaller, but more and more coherent subsets. For our purposes the selection of factors discriminating different failure behaviors (attributes used in the branch expressions) was of primary importance.

The data miner was instructed to provide a formula for determining the targets of illegal transitions in view of the circumstances (source state signature, original IP value, inverted bit etc.) in order to identify the regularities in failure phenomena. Leaves of the resulting decision tree were the target state signatures. The predicates assigned to the nodes along of a root–leaf path provide a deduction chain of the corresponding group (e.g. "if the code was generated by G and the source signature was S and the original IP value was between X and Y and the inverted bit was B then the target signature was usually T'"), which serves a behavioral description of the corresponding failure phenomenon.

For instance, a specific behavioral pattern occurring only in codes produced by one of the code generators was identified, thus indicating the extent of the influence of code generation to the faulty behavior.

It is important to highlight that although data mining delivers a behavioral description of the phenomena sufficient for the creation of a fault metamodel; their causal explanation (i.e. the discovery of the fault propagation chain) remains a task for an expert. In our case the verification of the high level fault model was performed by investigating the low-level implementation.

3. Conclusions

This paper outlined a method and environment for constructing fault metamodels applicable even in early system design phases to characterize the behavior of applications in presence of faults. The measurement results collected while executing fault injection campaigns on benchmark applications were analyzed by an intelligent data miner. Non-dominant, rarely occurring failure modes were identified by the analysis. The level of dependence of the failure phenomena from the implementation technology was estimated. A statechart-level fault model was developed based on the phenomena discovered.

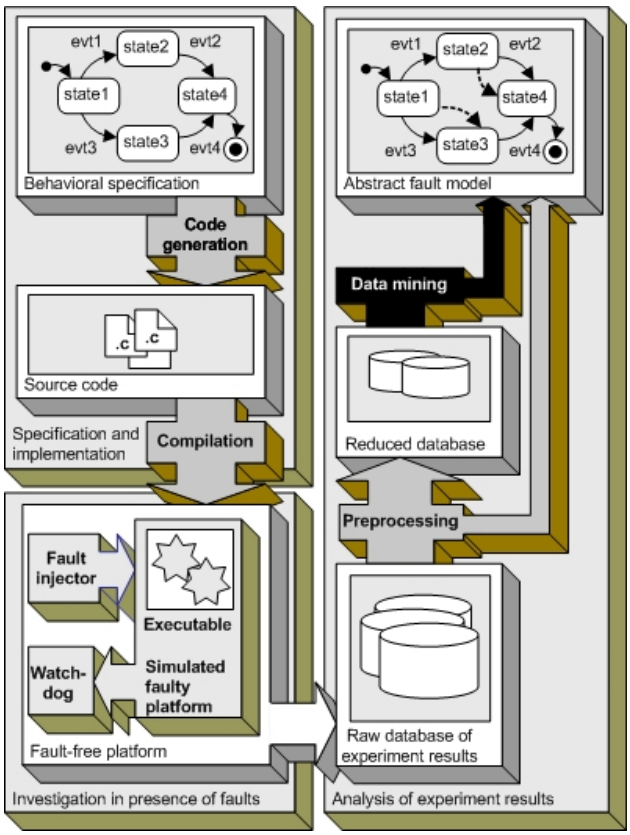

Figure 1. Overview of the method

References

[1] A. Pataricza and G. Pintér "Data Mining in Fault Injection", In Proc. DDECS 2003.
[2] I. Majzik, J. Jávorszky, A. Pataricza and E. Selényi "Concurrent Error Detection of Program Execution Based on Statechart Specification", In Proc. EWDC-10. 1999.
[3] IBM. "Intelligent Miner for Data. Applications Guide" 1999.