DATA MINING IN FAULT INJECTION *

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

Abstract. *This paper outlines a method and environment for reflecting the effects of faults in the behavioral software models providing a way for building abstract reusable fault models that are applicable even in early phases of the software development. The main focus is on the application of intelligent data processing methods for extracting important phenomena from the database of fault injection experiments.*

1 Introduction

Recent design for dependability techniques rely more and more on model-based verification and on the use of dependability design patterns (e.g. modular replication) for both the hardware and software parts of the design. Although high-level redundancy based design for dependability patterns provide good fault tolerance by the principle, they are not applicable in cases where the overhead caused by the high degree of redundancy cannot be tolerated like in the majority of non-mission critical embedded systems. In these cases an exact fault model has to be developed and fault tolerance measures have to be applied that exactly fit the fault model without introducing unacceptable overhead.

Our approach is focused on the intelligent processing of measurement results collected while investigating the behavior of the tested application during fault injection. The goal is the identification of behavioral patterns that can be used for building high-level fault models.

2 Overview of the approach

The *behavioral model* in our approach was described by a UML statechart that served as a basis for automatic code generation and as reference information for failure detection as well.

The experiments were performed on a *simulated faulty platform* developed by us consisting of a software-implemented fault injector and a statechart-level control flow checking mechanism [2]. The tested programs were instrumented to send signatures assigned to states of the UML statechart. Deviations from the behavior specified by the statechart were detected and stored in a database by a watchdog processor. The database can be seen as a log file, where the entries indicate the crash of the tested program or the detection of a state

*This work was supported by the project OTKA T-030804 of the Hungarian Scientific Research Fund. A part of the software tools was provided in the framework of IBM University Scholar Program.

transition that is not allowed by the statechart. These results are coupled with the exact circumstances of the fault injection (inverted bit, original register value, etc.). In the pilot experiment 200000 fault injection runs were performed by injecting single bit inversions in the Instruction Pointer (IP) register during the run of a simple benchmark application.

The most interesting phase of processing the experiment results was the one where *data mining* methods were applied for extracting important phenomena from the database of fault injection experiments. In our case classification, a special kind of predictive modeling implemented in the IBM Intelligent Miner for Data [1] was used. The method aims at providing a function for determining the value of an attribute (record field) in view of the other ones. This goal is achieved by discovering the correlation between the attributes.

The result of the analysis is presented in a form of a decision tree. The branches of the tree are labeled by logical formulae over the attributes separating the actual data set to two subsets, the subset that satisfies the formula and the one that does not. The root of the tree represents the entire database, the leaves correspond to specific values of the attribute to be determined. For our purposes the selection of important factors (attributes used in the branch expressions) was of primary importance.

In the pilot experiment we tried to identify rarely occurring complex phenomena. We were searching for a pattern of illegal state transitions (i.e. the high-level manifestation of a low-level fault) in the database. Identification of a pattern like this can be used for constructing an abstract statechart-level fault model for reflecting the effects of a fault class. In our case the data miner was instructed to provide a formula for determining the targets of illegal transitions in view of the circumstances (inverted bit, original register value, source state signature etc.). Leaves of the resulting decision tree were the target state signatures, the internal nodes were predicates over other attributes dividing the actual data set into partitions building this way root–leaf deduction chains (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”).

We were able to identify a behavioral pattern that was typical to codes produced by one of the code generators when inverting a specific bit in the IP register. We were able to clearly explain the effect chain that resulted in the phenomenon highlighted by the data miner. This way the data mining was shown to be applicable for identifying rarely occurring but typical phenomena that would be suppressed otherwise in the large amount of measurement results.

3 Conclusions

In the paper we illustrated the potential of intelligent data analysis in the experimental dependability evaluation focusing on automated feature recognition methods that can help in identifying effects caused by faults including rare ones. This approach enables the construction of abstract, reusable fault models that are applicable even in early phases of development (e.g. for designing fault tolerance based on forward recovery).

References

[1] IBM. *Intelligent Miner for Data, Applications Guide*. 1999.

[2] I. Majzik, J. Jávorszky, A. Pataricza, and E. Selényi. Concurrent Error Detection of Program Execution Based on Statechart Specification. In *EWDC-10*, Vienna, Austria, 1999.