Testing the Resilience of Fail-Operational Systems Early On with Non-Intrusive Data Seeding

Joachim Fröhlich
Siemens AG
Otto Hahn Ring 6
81739 München, Germany

Jelena Frtunikj
fortiss GmbH
Guerickestrasse 25
80805 München, Germany

Alois Knoll
Technische Universität München
Boltzmannstrasse 3
85748 Garching, Germany

I. Fail-Operational Systems are Resilient

Fail-operational systems are resilient systems. Fault patterns and the system structure, e.g., the degree of redundancy, the independence of fault regions and the availability of resources, determine the elasticity of a fail-operational system. Faults can be considered to deform a fail-operational system temporarily or permanently. Faults that occur temporarily or intermittently give systems reversible resilience. Permanent faults and component failures give systems irreversible resilience. When working under real-time constraints, the pressure on a fail-operational system increases. Mechanisms for detection, evaluation and handling of faults must attempt to promptly reshape the system when deformations occur.

II. Hard and Soft System Deformations

The safety analysis and the corresponding fault model of a fail-operational system define the HW and SW faults that it must detect and handle. HW faults, however, do not occur deterministically; they arise stochastically, which can be problematic for new HW for which there is little empirical data. SW faults, on the other hand, occur with certainty. The problem with SW faults is to find the faulty program execution paths and critical combinations of program state and data for complex systems in different situations. Under these conditions, design, implementation and execution of repeatable tests that stimulate the system with faults (fault-injection tests) are difficult and expensive. Tests are considerably facilitated if they can access system internals, that is, can read and write data as it flows through the system as well as access data quality indicators (time and value quality).

III. Tests in Virtual and Real Environments

From technical and economic points of view it is advisable to start as early as possible with the evaluation of individual and integrated systems. It is profitable and advisable to test their capabilities to handle faults promptly and correctly even prior to the availability of actual system HW through the use of virtual environments like HW abstracting test beds. Hence, tests evaluating resilience properties of a system have to be designed and maintained as cross-platform and cross-phase regression tests. To this end, a safety-critical system must be modular and portable. In particular, a fail-operational, real-time system must enable fault-injection tests free of side-effects in order to avoid undesirable functional and temporal distortions of the system under test (SUT). Only then can these tests provide reliable statements about the fault-elasticity of a system that are traceable between virtual and real environments.

IV. Functions and Form of Testable Systems

To obtain definite statements about the effectiveness of fault handling mechanisms from test runs, tests must seed data, including HW and SW faults, at precise time points and system locations. More specifically, the SUT must enable non-intrusive monitoring and manipulation of signal data, state data and data quality indicators. We therefore assume the SUT to basically have the following properties (Fig. 1):

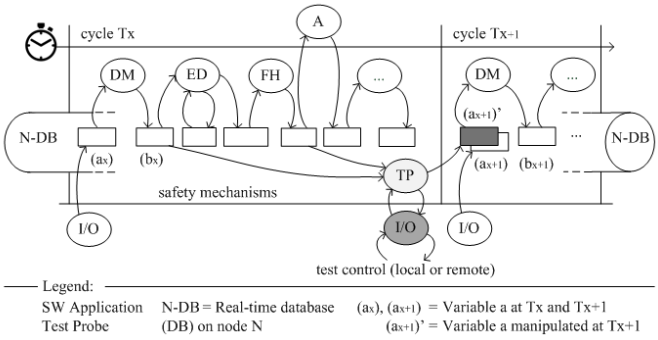

Fig. 1. Data flow in a time-triggered system node with a built-in test probe

(1) *Time-triggered architectures* (TTA) [3] behave deterministically because systems control events and not vice versa as in event-triggered architectures. In each cycle the system sequentially executes safety operations, like data monitoring (DM), error detection (ED) and fault handling (FH).

(2) *Databases* continually capture, for each node and cycle, flows of signal data, state data and data quality indicators.

(3) All system nodes contain *built-in test probes* (Fig. 1, Fig. 2, TP). TP operations are always scheduled at the end of each cycle. In this position between adjacent cycles, a TP can (i) monitor data accumulated in the N-DB during the last cycle and (ii) manipulate data for the next cycle.

(4) TPs use exclusively reserved resources, including time slots, memory areas and communication links. Other modules cannot use TP resources, even when a TP is deactivated; otherwise, a TP would be intrusive because the SUT would behave differently from the final system.

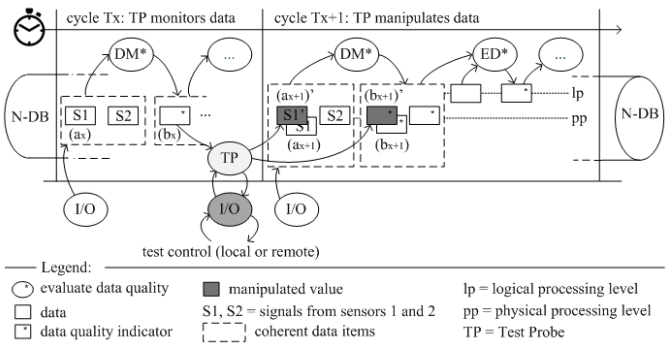

Fig. 2. Manipulate data (ax+1)' or data quality indicator (bx+1)'

V. System of Non-Intrusive Test Probes

Remote tests, local tests, or both combined, control a test probe (TP). Remote tests require a point-to-multipoint connection between one common, central test processor at one end and several test probes at the other ends. Test programs on the central test processor monitor and control nodes system-wide, while test programs on a local node operate within the node. Probe instructions are location transparent: a test probe does not know where the program that issues instructions runs.

Probe instructions manipulate, monitor and check data of the probe-containing node with a single cycle delay. If a remote test program on a central test processor controls one or more test probes, then the test reaction delay for a round trip of a test control loop takes 4 cycles minimum: (i) Test probes read and send values to the central processor in cycle Tx, (ii) the central processor evaluates received values, takes decisions and instructs test probes in cycle Tx+1, (iii) test probes follow the instructions received in cycle Tx+2 which are (iv) effective 1 cycle later in cycle Tx+3. Throughout a test, probes and the central test processor monitor node vitality and test plausibility. Because of test probes running synchronously with the node, tests provide cycle-accurate results in virtual environments and real environments for target HW in the lab and in the field.

VI. Testing System Resilience

For demonstration purposes we test a system of three nodes, instances of which are parts of larger systems that can be built with RACE[1] [1], [4]. Two redundant sensor nodes in the system periphery provide signal data to a central processing node. The test (Algorithm 1 written in ALFHA[2] [2]) checks the elasticity of the central node if a sensor fails temporarily (reversible resilience) or permanently (irreversible resilience).

The test injects (seeds) faults into one of the input channels that connects the central node with the sensors (Fig. 2, (ax+1)') after all nodes are up and run normally (lines 11-12). Alternatively, the test can intervene in the data flow in the central node by manipulating data quality indicators that signify value and time quality of the sensor data (Fig. 2, (bx+1)'). Algorithm 1 allows both approaches (line 18) because the location where the test seeds data is a parameter, as is the fault duration (lines

[1]Robust and reliant Automotive Computing environment for future Ecars, www.projekt-race.de/en

[2]Assertion Language for Fault-Hypothesis Arguments

Algorithm 1 Masking sensor fault on signal receiver

```
1:  TEST Masking sensor fault WITH
2:      N, N1, N2, // Central node N process signals from sensors N1, N2
3:      F, // N executes system function F
4:      L, V, // Location L to inject V in N
5:      C, D, // Injection instant (cycle C) and duration (number of cycles D)
6:      T, // Tolerance across 2 succeeding values V
7:      P // Length of node period P in milliseconds
8:  EXPECT Function gets data from redundant sensor
9:  SYSTEM PERIOD P // Cycle length
10: TIME BOUND 1000 // Obtain definite verdicts within 1000 ms
11: SETUP Masking sensor fault WITH N, N1, N2
12: CLOCK WHEN N*.State == eNormalOperation
13: INVARIANT // N, F are resilient to sensor shocks:
14:     // Uninterrupted, smooth data stream
15:     N.F.Input == N.F.Input@[-1] TOLERANCE T
16: BEGIN
17:     // Invariant holds (!!) during fault injection ...
18:     [ C : < +D ] !! N.L = V // ... from C to C+D-1
19: END
```

4-5). In any case, the system function that processes the sensor data must always get valid input values (test invariant, line 15). It is not necessary for the test to simulate the environment because, with RACE, nodes can start and run in a neutral mode processing default values. With the steering wheel in neutral position (default), the reversible elasticity of a node executing the steering function can be tested as follows:

N = $CentralNode$, $N1$ = $Sensor1$, $N2$ = $Sensor2$,
F = $Steering$, L = $In.Sensor1$, V = $InvalidData$,
C = 100, D = 2, T = 1, P = 10

For testing irreversible elasticity, we extend the fault injection duration *D* from 2 cycles to, say, 1000 cycles indicating a permanent failure of *Sensor1*. For testing the fault-resilience of the system at various nodes, including the communication links between them, test invariant (line 15) and fault injection (line 18) must refer to different nodes. Then we can change the fault-injection location to, for example, *Sensor2.Out.Steering*, independent of the test invariant still checking the correctness of the data stream at *CentralNode.Steering.Input*.

VII. Resilience Tests Strengthen Safety Cases

Testing fail-operational systems early on with non-intrusive data seeding provides quick and precise feedback in virtual and real environments. Such reliable tests are strong arguments in safety cases of fault-resilient, fail-operational systems.

References

[1] J. Fröhlich and R. Schmid. Architecture for a Hard-Real-Time System Enabling Non-intrusive Tests. In *25th IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, page 24, November 2014.

[2] J. Frtunikj, J. Fröhlich, and A. Knoll. Qualitative Evaluation of Fault Hypotheses with Non-Intrusive Fault Injection. In *5th International Workshop on Software Certification (WoSoCer 2015)*. IEEE, November 2015.

[3] H. Kopetz and G. Bauer. The Time-triggered Architecture. *Proceedings of the IEEE*, 91(1):112–126, January 2003.

[4] S. Sommer, A. Camek, K. Becker, C. Buckl, A. Zirkler, L. Fiege, M. Armbruster, G. Spiegelberg, and A. Knoll. RACE: A Centralized Platform Computer Based Architecture for Automotive Applications. In *Vehicular Electronics Conference and the International Electric Vehicle Conference (VEC/IEVC)*. IEEE, October 2013.