Test Automation for Avionic Systems and Space Technology (Extended Abstract)

Jan Peleska
Universität Bremen, Technologiezentrum Informatik TZI, jp@tzi.org
Cornelia Zahlten
Verified Systems International GmbH, Bremen, cmz@verified.de

Introduction. In this article, we give an overview about typical objectives and problems to be encountered when testing embedded systems in avionics or space technology. The overview is based on experiences gained during (1) the verification and test of the DMS-R FTC fault-tolerant computer system developed by Daimler-Chrysler Aerospace for the International Space Station ISS (see [6, 1]), (2) the test of the PTC Power and Thermal Controller for the ABRIXAS satellite developed by OHB and (3) an automated test system currently developed for the test of the Airbus CIDS Cabin Intercommunication Data System Family developed by Daimler-Chrysler Aerospace Airbus, (K.I.D.-Systeme). In these projects, automated tests were performed using the RT Tester tool (formerly called VVT-RT) developed by Verified Systems International in collaboration with the TZI. A major part of the underlying theory has been published in [3, 4, 5].

Hardware in the Loop Tests. Standards for computerised avionics and space systems [2] require tests on unit level, integration tests and tests on system level. The effort required depends on the criticality of the system under consideration. In any case, special emphasis is put on system tests, since the frequent use of specialised hardware components, non-standard interfaces and rigid timing conditions require a thorough analysis of the cooperation between software and hardware components. The reactive nature of embedded avionics and space systems leads to test specifications describing the expected behaviour with respect to data transformations, sequencing of inputs and outputs and I/O timing. Timing requirements are often too fine-grained to be tested in a manual way. Moreover, embedded controllers often interface to sensors and actuators handling time-continuous data in addition to digital message interfaces and control switches. As a consequence, the most general test suites require hybrid real-time test specifications and *Hardware-in-the-Loop Tests* where separate computers act as test drivers controlling the data flow to/from the system under test (SUT) in real-time. A generic test configuration is depicted in Figure 1. The test drivers replace the operational environment of the SUT and access the SUT via system interfaces and additional monitoring and test control channels. For the *RT Tester* tool, the problem of accessing non-standard interfaces (for example, ARINC 429/629 and MIL-STD 1553 busses) is solved by using a distributed architecture where *Interface Modules* (hardware plus software) directly connect to the SUT and implement an abstraction of the concrete interface. The *Abstract Machines* controlling test generation, execution and evaluation only have to handle abstract values to be passed from/to interface modules via standard interfaces (e. g., Ethernet, shared memory).

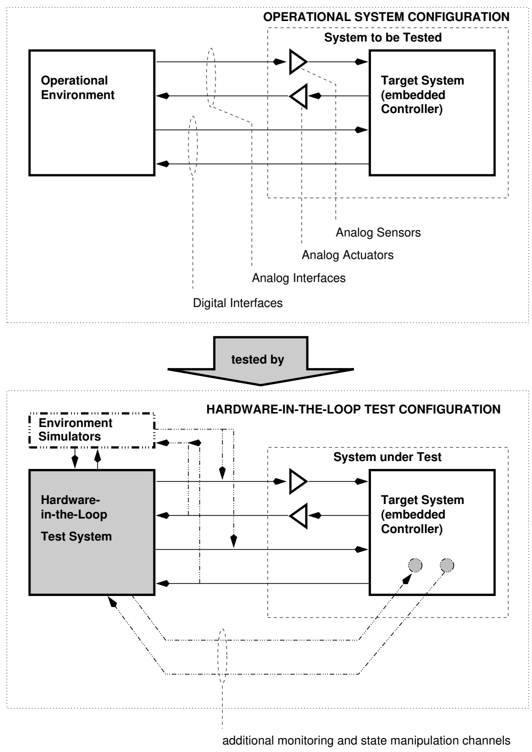

Figure 1: Generic hardware-in-the-loop test configuration.

Undefined Application Layers. To reduce the costs for aircraft or space station/space ship development, there is a growing tendency to use standardised computers where different application layers (e. g., software controlling different experiments in a space station) can be integrated. In general, the computer manufacturers design and develop the computer hardware, operating system software and a re-usable application program interface (API) providing a set of resource control and communication services and optionally fault-tolerance or safety services to the application layer. Testing such a computer for the manufacturer requires to design a “most general” *Test Application Layer (TAL)*, such that a successful test suite with this TAL implies that tests would also succeed for more specific application layers, as long as these respect the rules defined in the API specification. Conceptually – though physically residing inside the computer hardware – the TAL is a part of the SUT's test environment; the SUT now consists of the hardware, the operating system and the API. In the *Timed Trace Model of CSP*, for example, this problem can be solved by means of the following

Theorem. Let *specSUT* be an abstract specification process of the desired SUT behaviour. Let *impSUT* denote the SUT implementation and *ENV* the environment (including the TAL) used for testing. Then, if $(ENV \| specSUT) \sqsubseteq_T (ENV \| impSUT)$ and $ENV \sqsubseteq_T ENV'$ holds, $(ENV' \| specSUT) \sqsubseteq_T (ENV' \| impSUT)$ follows.

This theorem has the following implication: If we design the TAL in such a way that every specific application layer will be a timed trace refinement of this TAL (i. e., $ENV \sqsubseteq_T ENV'$), then successful test execution with the TAL (observe that successful exhaustive tests imply $(ENV \| specSUT) \sqsubseteq_T (ENV \| impSUT)$) guarantees that tests with each specific application layer would be successful, too.

For the RT Tester tool, we have developed a strategy how to derive most general TALs from API specifications. The TAL behaviour desired for a specific test step can be controlled from the abstract machines residing on the test driver computers. To this end, either additional control channels are used, or control commands are transmitted using the user data communication channels of the SUT.

Generic Test Specifications. Another widely-used approach to reduce development costs is the design of *Data-Driven Functionality*, where major parts of the functional behaviour are not hard-coded in the software, but can be defined by means of tables which are loaded at system initialisation, containing "software switches" and other parameters. This may be regarded as an object-oriented approach, where the software code defines generic objects to be instantiated at system initialisation using the tables. The implications on testing are twofold: First, test coverage strategies must take into account different table definitions. Second, the test specification should allow for generic parameters, since otherwise each new table definition to be used in a test execution would require a new test specification. To address the second problem in the RT Tester tool, we use timed CSP specifications with *Parameterised Process Specifications*, the parameters containing either local state information of the process or generic values to be instantiated at the begin of each test execution. The specifications are structured in such a way that the concrete values for generic parameters are read from the table data via initialisation channels.

For example, the CSP process *NS_AUTO1* below describes a simplified version of the decision procedure for automatic activation of the *No Smoking Signs* in the Airbus A330/340 family: The decision depends on (1) the state of the landing gears, (2) the state of the slats and (3) a generic software switch *nsa* specifying the dependency on (1) and (2). If *nsa* = *nsSolII*, the decision (communicated via channel *ns_auto_cond*) depends on both landing gears and slats, otherwise it depends on the state of the landing gears only. The generic parameter is instantiated in process *NS_AUTO* via initialisation channel *init_nsa*.

```
NS_AUTO =
  init_nsa?nsa
  -> NS_AUTO1(nsa,downCompressed,passive)

NS_AUTO1(nsa,ldg,slats) =
   (if ((nsa == nsSolI and ldg == downLocked)
         or (nsa == nsSolII
             and (ldg == downLocked or slats == active)))
    then (ns_auto_cond!true -> NS_AUTO2(nsa,ldg,slats))
    else STOP)
    []
    landing_gear_control_unit?ldg -> NS_AUTO1(nsa,ldg,slats)
    []
    slats_computer?slats -> NS_AUTO1(nsa,ldg,slats)
```

Conclusion. We have described a number of objectives and problems which we regard as characteristic when testing embedded avionic and space systems. Additional aspects which cannot be described in detail in this article are: (1) Abstraction/refinement of concrete interface data (we follow an approach based on *Action Refinement*), (2) specific strategies for the test of fault-tolerance properties, (3) separation of test objectives containing correctness of data items, control commands and time-continuous values and (4) use of *Virtual Reality Techniques* allowing application experts to develop test specifications without detailed knowledge about formal specifications.

References

[1] Bettina Buth, Michel Kouvaras, Jan Peleska and Hui Shi: Deadlock Analysis for a Fault-Tolerant System In Michael Johnson (Ed.): Algebraic Methodology and Software Technology. Proceedings of the AMAST'97, Sidney, Australia, December 1997, Springer LNCS 1349 (1997), 60-75.

[2] RTCA DO178B: *Development considerations in airborne computer systems.* (1993).

[3] J. Peleska: Formal Methods and the Development of Dependable Systems. Habilitationsschrift, Bericht Nr. 9612, Dezember 1996, Institut für Informatik und praktische Mathematik, Christian-Albrechts-Universität Kiel (1997).

[4] J. Peleska and M. Siegel: Test Automation of Safety-Critical Reactive Systems. *South African Computer Jounal* (1997)19:53-77.

[5] J. Peleska: Testing Reactive Real-Time Systems. Tutorial, held at the FTRTFT '98. Danmark Technical University, Lyngby (1998).

[6] Gerd Urban, Hans-Joachim Kolinowitz and Jan Peleska: A Survivable Avionics System for Space Applications. Published in Proceedings of the FTCS-28, 28th Annual Symposium on Fault-Tolerant Computing, Munich, June 23-25, 1998, 372-381.