Automated Test Suites for Modern Aircraft Controllers

Jan Peleska
Technologie-Zentrum Informatik TZI
Universität Bremen, P.O. Box 330440, D-28334 Bremen,
e-mail jp@tzi.de

14th January 2003

Abstract

Integrated Modular Avionics (IMA) is a unifying concept for architecture, operating system API and configuration of aircraft control systems. Controllers developed according to this concept are tested in two phases: The test objective of phase one is to establish compliance of hardware, operating system and configuration for a specific purpose. In phase two, compliance of the application software as integrated in the IMA controller is checked. In this article, aspects of automated test suite generation for phase one are discussed. Examples are derived from ongoing work of the author's research group currently performed for Airbus Deutschland in the context of hardware-in-the-loop tests for Airbus A380 aircraft controllers.

1 Introduction

"Modern" Aircraft Controllers: Integrated Modular Avionics. The technical evolution of Airbus aircraft technology has led to two important innovation steps related to avionic software and control computers, first to be applied in the Airbus A380 currently under development:

- The ARINC 429 Bus technology used up to now as the most important means for data exchange between aircraft controllers will be replaced by the *Avionics Full-DupleX Switched Ethernet (AFDX)*.
- Instead of customised proprietary controller hardware and operating system software, a major subset of the A380 controllers will be based on *Integrated Modular Avionics (IMA)* components, where different applications run on standardised hardware, using the same distributed operating system. The operating systems presents a set of well-defined standardised (see [1]) services to the software application layer. These services are summarised in the *Airbus Application Program Interface (API)*.

Each IMA module (i.e. a single controller developed according to IMA technology) consists of standardised (single) processor board, aircraft connector, back-plane, interface bus and interface boards. An AFDX interface is always provided for communication with other aircraft controllers. As required by the applications to be allocated on the IMA module, I/O boards for

- ARINC 429,

- CAN,
- Digital I/Os with various electrical specifications,
- Analogue,
- RS232

interfaces are configurable according to the application requirements. The Airbus API provides a unified channel abstraction, so that application software may access these interfaces using a standardised port concept which is independent on the physical characteristics of the underlying I/O device.

The IMA operating system supports *partitioning* for groups of software tasks: Tasks of one partition may access data managed by tasks residing in other partitions only via well-defined communication channels. The partitioning technology supports *fault containment*: Faults – e. g. illegal memory access – occurring in one partition do not affect tasks operating in others.

Strategy for IMA Verification, Validation and Testing. The re-use effect gained by IMA technology during the development phases has an important impact on the verification, validation and testing (VVT) process: Controller hardware and operating system may now be provided by manufacturers different from the companies developing avionics application software. As a consequence, a major portion of the HW/SW integration process and its related VVT activities will have to be performed as a joint activity of IMA module manufacturer and avionics software developers, or these activities will be shifted to the system integrators.

As a consequence, a novel testing strategy has been elaborated by the responsible Airbus authorities and their partner companies. The complete VVT process will be structured into several steps:

1. The main objectives of *IMA Module Verification Step 1* focus on one IMA Module without an application-specific configuration and – consequently – without any application software:
    - Demonstration of correct operating system behaviour of one IMA Module as far as visible at the Airbus API.
    - Demonstration of partitioning (robustness of IMA partitions against faults occurring in other partitions).
    - Demonstration of I/O functionality.
    - Demonstration of correct behaviour for arbitrary configurations compatible with the underlying CPU and I/O hardware.
2. The main objective of *IMA Module Verification Step 2* is to verify an application specific *module configuration*: Among other properties, the IMA configuration table specifies the admissible data flow between
    - tasks residing in one partition,
    - tasks residing in different partitions (also across IMA module boundaries),
    - tasks and I/O interfaces.

3. The main objective of *IMA Module Verification Step 3* is to verify that a network of configured IMA modules supports distributed inter task communication via AFDX in compliance with the application-specific configuration tables. Observe that at this point in time only the configuration tables are application-specific – the "real" application layer still remains to be integrated.

4. *IMA Module Verification Step 4* finally demonstrates the correct operation of the integrated application software.

IMA Testing Approach – TZI Responsibilities. The author's research group at the TZI currently develops – in collaboration with Airbus Deutschland and with the TZI spinoff Verified Systems International GmbH – concepts, hardware, software and test specifications for the test execution within the context of IMA module verification steps 1, 2 and 3. The present paper gives a description of some of the main concepts of our approach.

Background. Since 1998, Verified Systems and the TZI provided various services and products for Airbus Deutschland and Airbus partners in the field of

- Introduction of new testing methods, strategies and test system architectures,
- Tools (software and hardware) for automated test generation, execution and evaluation,
- Test specification development,
- Execution, evaluation and documentation as well as error analysis for concrete tests.

They have been applied to various controllers of the Airbus aircraft families A330/340, A340-500/600 and A318. The core of all our test automation tool components is the RT-Tester test automation system [19] developed by the author's research group in collaboration with Verified Systems.

Related Work. From the theoretical point of view, the problems of automatic test generation, test execution in hard real-time and test evaluation have been investigated by the author and his research team at Bremen University in collaboration with several other scientists. In [9, 11, 12] the theory for automated testing of reactive systems without timing requirements are described. The foundations of the theory are based on deNicola's and Hennessy's testing equivalence elaborated for process algebras with acceptance tree semantics [4]. For real-time testing, our formalisms are based on the semantics of Timed CSP (Communicating Sequential Processes with Time, TCSP) as given by Schneider [21]. Comprehensive introductions to CSP are given in [6, 18]. For the associated algorithms implemented in the RT-Tester tool we made use of a theorem about the executability of CSP specifications in hard real-time, which has been formulated by the author [13] and established in complete form by Oliver Meyer [7]. Additional material with respect to testing avionics systems can be found in [15, 16, 17]. The test application concept described in Section 3 has been elaborated and applied for tests of a fault-tolerant computer developed by Astrium for the International Space Station [14].

The field of test automation based on formal methods is currently investigated world-wide by several research groups. We name [3, 8, 20, 22, 2] as a set of representative publications which also give an extensive overview of existing publications in this research area.

Overview. In Section 2, the organisational model of the testing environment is described. Section 3 describes the utilisation of generic test applications which are integrated in place of "real" applications within the IMA modules. In Section 4, the development of generic test specifications – to be instantiated from concrete IMA module configuration tables – is explained. Finally, Section 5 presents the overall test configuration as it is currently developed by Verified Systems in collaboration with the author's research group at the TZI.

2 Organisation Model of the Testing Environment

Generic RT-Tester Environment. The IMA testing environment may be regarded as an instantiation of the generic RT-Tester test automation system – tShe organisational model of the latter is shown in Figure 1. An RT-tester engine is structured into 3 conceptual layers:

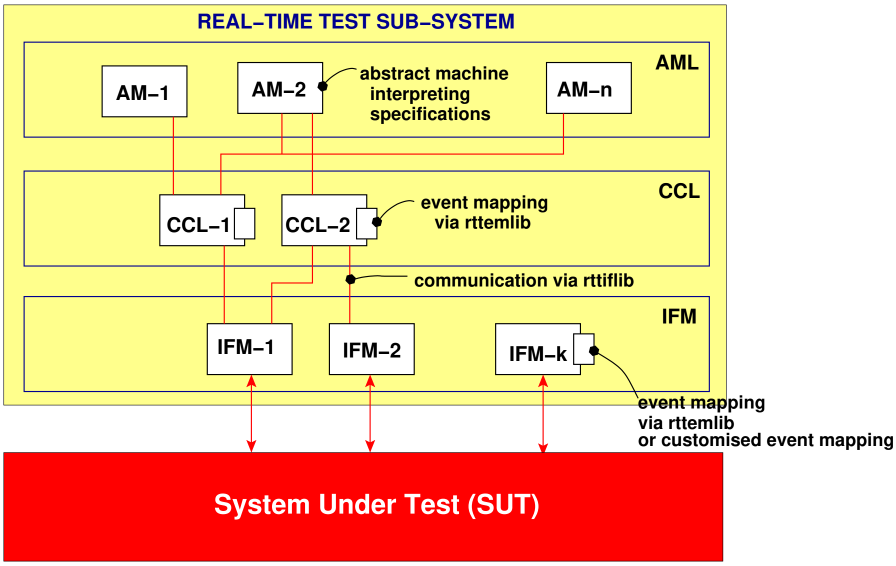

Figure 1: Organisational model of the RT-Tester system.

- The *abstract machine layer* consists of abstract machines (AMs) interpreting pre-compiled executable test specifications in hard real-time while running in parallel during a test execution. AMs implement algorithms for automatic on-the-fly generation of test data and automatic checking of the *System Under Test (SUT)*'s responses to the stimuli generated by the testing environment. Interfaces are denoted as *channels* encoded on an abstract level which is independent on the concrete software or hardware interfaces used in order to connect to a concrete SUT.

- The *interface module layer (IFML)* consists of interface modules (IFMs) implementing the *interface refinement* from abstract abstract machine interface channels to concrete SUT interfaces and the *interface abstraction* by mapping concrete SUT outputs to abstract AM channels.

- RT-Tester supports a many-to-many interface channel concept: Different abstract machines may generate output on the same channel c. These outputs are mapped by the *communication control layer* to all IFMs and AMs having declared c as an input channel.

The organisational model described above has been motivated by the following considerations: Testing embedded systems requires *parallel* processes for simulation of external systems in the SUT operational environment, stimulation of specific events in order to control the test execution and checking of SUT responses to the SUT inputs given by the testing environment. The simple sequential test scripts consisting of input commands to the SUT followed by checking expected SUT outputs are inappropriate for more complex applications where several logical threads cooperate in parallel. In these situations, the expected SUT behaviour depends on several state variables which may change independently due to asynchronous stimulations on different interfaces. Outputs e of one abstract machine are often consumed not only by the SUT but also by other AMs whose future behaviour or whose checking results depend on the occurrence of e. If an SUT interface changes or when re-using software integration test specifications on HW/SW integration test level it is practical just to exchange the related interface modules without having to change the test specifications as well. This can be accomplished by the interface abstraction concept.

Organisational Model of the IMA System Integration Test Bench. In the context of IMA module testing, the RT-Tester system is instantiated and extended to the organsiational model of the *IMA system integration test bench (SIB)* depicted in Figure 2

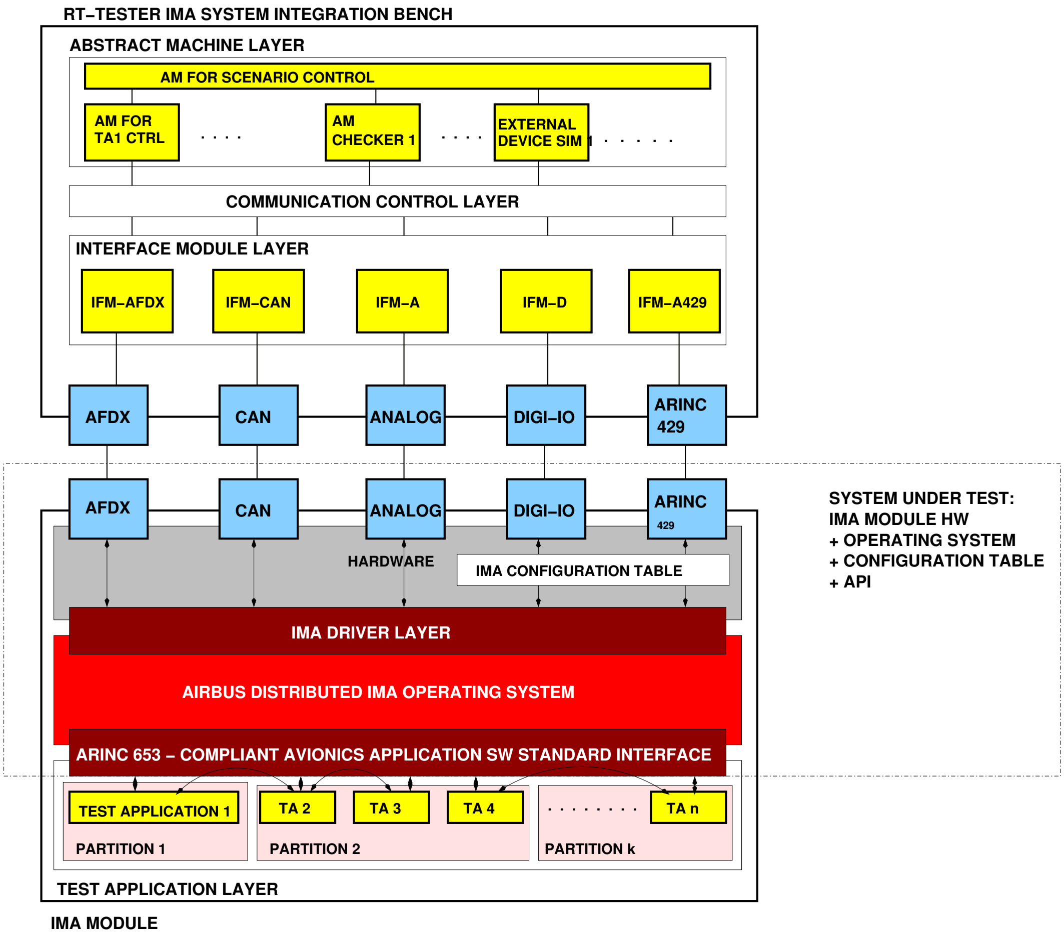

Figure 2: Organisational model of the IMA system integration test bench.

As indicated in the introduction, the system under test consists of

- IMA module hardware, including I/O interfaces,
- The IMA configuration table (test-specific creation in test step 1, application-specific creation for steps 2, 3, . . .),

- The distributed IMA operating system,
- The Airbus API.

Since a concrete application layer is missing during the test steps under consideration, a *test application layer (TAL)* is allocated as a collection of pseudo application tasks in one or – depending on the configuration used in the specific test – several partitions of the IMA application layer. Though allocated on the IMA itself, the TAL is considered as a part of the testing environment: As will be described in more detail in Section 3, the TAL acts as a set of abstract machines "remote-controlled" by the AMs residing on the test engine.

The concrete interface modules required for the IMA SIB act as adapters to the AFDX, CAN, ARINC 429 and various analogue and digital I/O interfaces.

Abstract machines are organised into 4 groups:

- Each AM of the first group acts as *TA controller* for one test application (TA) residing in the IMA Module, see Section 3 for further details.
- The AMs of the second group act as external *simulators* for CAN, digital, analogue and ARINC 429 inputs to the SUT. They act as slaves to the controller AMs of group 1. For example, a simulator may generate analogue inputs to the SUT following a time and slope specification received from a TA controller AM.
- Group 3 consists of AMs acting as *checkers (test oracles)*: They encode expected results specifications and check the compliance of SUT outputs in relation to the inputs generated by AMs of group 1 and 2. Apart from errors, warnings and requirements tracing messages, checkers to do not generate any outputs; they only monitor events generated by the SUT and by other AMs.
- Group 4 consists of high-level abstract machines acting as *scenario controllers*: Test executions are often structured in to different phases, e. g., time intervals where AMs simulate only normal behaviour or also exceptional behaviour in the SUT environment. The high-level events generated by scenario controllers are consumed by the AMs of groups 1, 2 and 3. They are never mapped to the SUT.

Data Flow During Test Executions. During test executions, AMs acting as TA controllers send AFDX messages to test applications in order to configure their behaviour. When receiving AFDX, CAN, analogue, digital or ARINC 429 inputs (generated by simulator AMs) on any of its configured ports, each TA will react according to its pre-programmed behaviour. The TA behaviour becomes visible at its output ports and is monitored by the checker AMs. Since the pre-programming has been monitored by the checkers as well, they can automatically evaluate observed TA behaviour against required behaviour.

3 Generic Test Applications

The motivation for the test application layer (TAL) is as follows: Suppose that a test execution succeeds for a set of parallel TAs. Then this execution will also succed for all "real" applications which are refinements of the corresponding TAs (a proof for this fact may be found in [14]). As a consequence, it is desirable to use test applications showing very large behavioural variations: This increases the confidence that real applications will be more restrictive – i.e., show less variations of behaviour – and therefore also operate correctly when embedded in the IMA application layer.

It would be infeasible to program different TAs for each "pseudo-application" and for the different relevant types of application behaviour. Instead, our concept is to program one generic TA which may be instantiated with the IMA configuration data. This allows for binding concrete I/O ports to generic ones during the instantiation process. However, this does not suffice to generate the required behavioural variations during test executions. To this end, we consider the TA as a *generic virtual machine*. Instantiation only binds concrete I/O ports. The concrete behaviour to be observed during a test execution is defined *on-the-fly* by means of behavioural control commands (called pre-programmed behaviour above) which are transmitted from abstract machines acting as TA controllers via AFDX channels during the test execution. The following types of commands may be given from the TA controller to its associated TA:

- Trigger Airbus API call with parameters specified by the behavioural control command (observe that returning data on an output port is also a parameterised API call),
- Assign variable values or literals to variables,
- Perform arithmetic operations on specified variables and literals,
- Execute hard-coded scenarios:
  - Generation of CPU maximum load or overload,
  - Generation of maximum load or overload on specific communication ports,
  - Illegal memory access (to verify fault containment).

The TA is implemented as a virtual *stack machine*; each behavioural control command is one of the following kind:

- `push [<literal> | <variable> | <reference>]`
- `execute <operation>`

`Push` commands result in putting the literal or the value of the referenced variable or its address onto an *operation stack* maintained by the TA. `Execute` commands result in calling the identified operation while popping the number of parameters it requires from the stack. Operation return codes are pushed back to the stack for further use by consecutive operations.

4 Generic Test Specifications

The main purpose of an IMA configuration table is to specify the available hardware interfaces as well as the channels connecting input or output ports of applications to corresponding ports of other tasks or to IMA interfaces. The distributed Airbus operating system supports one-to-many channels.

One of the main test objectives of an IMA test suite consists in the demonstration that the configured data flow may really be utilised in the specified manner. Therefore we instantiate a library of generic specifications for TA controllers, checkers and simulators with the concrete interfaces data derived from the IMA configuration table. To illustrate this concept consider the data flow graph shown in Figure 3: We distinguish 3 types of nodes representing physical interfaces, channels and (Test-) applications.The arrows represent the input/output ports

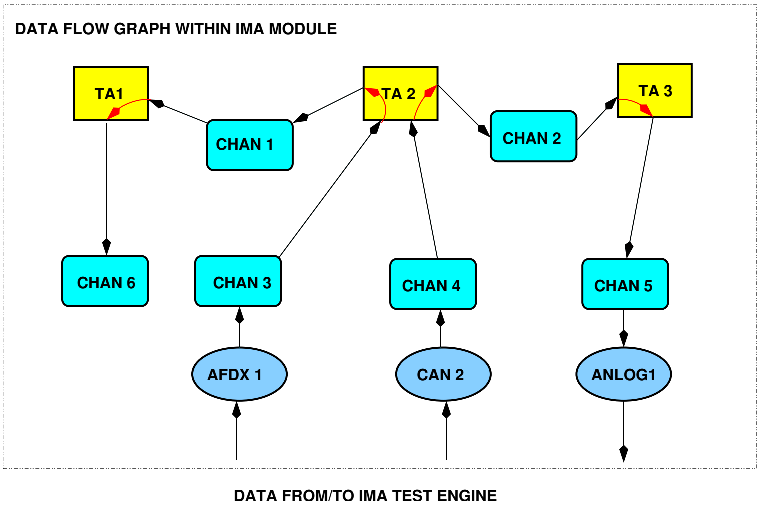

Figure 3: Data flow graph within the IMA module.

associated with the corresponding channels. An application may only access the configured channels, but it is free where to map (possibly transformed) input data. Therefore the controller abstract machines send different combinations of behavioural control commands to the TAs, each command specifying a routing and data transformation directive resulting in a "data flow switch" as indicated by the red arrows inside the TA boxes. Different TA controllers and interface simulators stimulate the IMA inputs in parallel, trying to cover all possible communication flows through the graph. The checkers compare inputs to the IMA module and observe IMA outputs, while checking the timing and data flow correctness. The checks are supported by *routing directives* encoded by TA controller AMs in their telegrams to the TAs: Using the stack machine commands sketched above, TA_2 will process the first part of a telegram received by its controller AM and then send the rest to another TA_1 using an inter-task communication channel. TA_1 continues interpreting the telegram until the complete set of stack machine commands has been processed.

5 Test Engine Architecture

The test engine part of the organisational IMA system integration bench model described in Section 2 is implemented on a *PC cluster architecture*, as sketched in Figure 4.

In an RT-Tester cluster, 2-CPU or 4-CPU PCs are connected via a switched high-speed data link (Myrinet, 2GBit/sec). A DMA-based interface developed for the communication control layer allows transparent communication of test events between abstract machines and interface modules residing on arbitrary cluster nodes. Communication cycles are highly deterministic with a resolution $t < 200\mu sec$.

In order to avoid interface bus bottlenecks (these are quite common "threats" when using conventional VME-based systems for real-time applications) the hardware interfaces are distributed over 3 PCI busses.

Hard real-time behaviour is enforced by utilisation of a Linux kernel enhanced by real-time system calls developed at the TZI: Abstract machines and interface modules are allocated as user threads on light weight processes (LWPs) operating on reserved CPUs without preemption by the Linux scheduler. User thread scheduling is managed within the RT-Tester system itself. Interrupts can be explicitly directed to the CPU which is most appropriate for their handling.

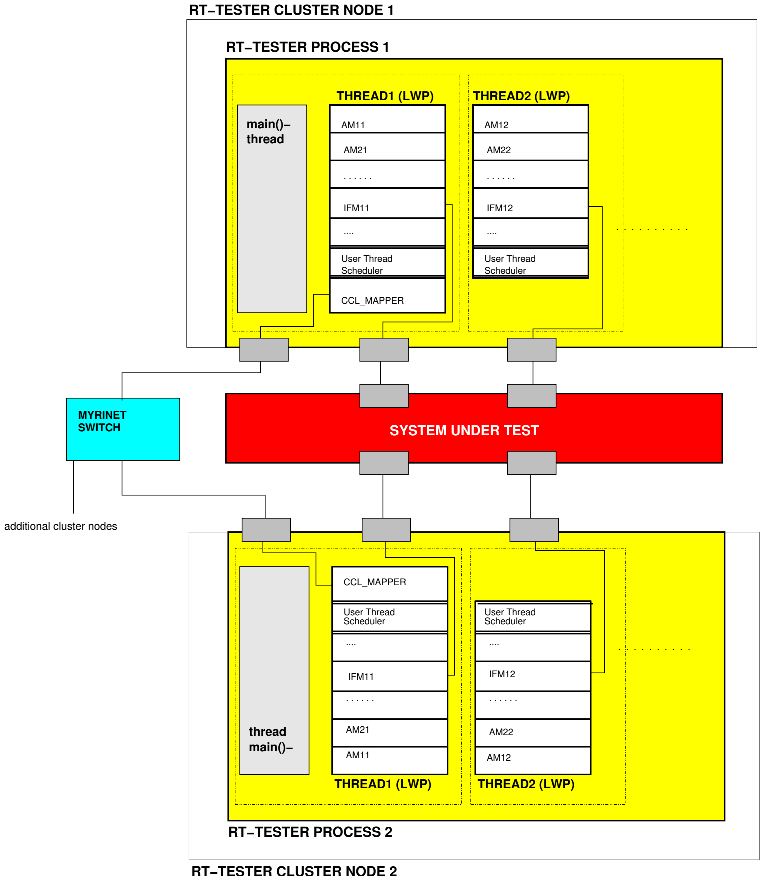

Figure 4: RT-Tester cluster configuration

6 Conclusion

In this article, we have presented a strategy, associated methods and tools for testing Integrated Modular Avionics controllers as used for the Airbus A380 aircraft. Our approach was based on a test application layer replacing "real" applications. This layer can be controlled from the test engine, so that all relevant possible application behaviours can be directly triggered by the testing environment. Since the IMA module behaviour depends on its configuration table, a generic library of test specifications can be instantiated using IMA configurations, each time resulting in a test execution behaviour and a specification of expected results which is compliant with the specific configuration. The test environment is implemented using a novel PC cluster technology offering deterministic timing behaviour and easy performance scalability.

Acknowledgments. A major part of the results investigated by the author and his research team at the University of Bremen and described in this article have been elaborated within the European VICTORIA research project focusing on new architectural, development and validation concepts integrating on-board functions for commercial airliners. Special thanks go to Stefan Bisanz, Dirk Meyer and Aliki Tsiolakis (TZI), Markus Dahlweid, Oliver Meyer and Cornelia Zahlten (Verified Systems) and Josef Kruse (Airbus Deutschland) for stimulating discussions and excellent collaboration. Last, but not least, the author would like to express his thanks to the members of the committee for the workshop *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen* for their invitation to present this article.

References

[1] Airlines Electronic Engineering Committee (1997). *ARINC 653: Avionics Application Software Standard Interface*. Aeronautical Radio, Inc. 2551 Riva Road, Annpolis, Maryland 21401

[2] Brinksma, Ed and Tretmans, Jan. (2000) Testing transition systems: An annotated bibliography. In F. Cassez, C. Jard, B. Rozoy, and M. Ryan, editors, *Proceedings of Summer School MOVEP'2k Modelling and Verification of Parallel Processes, Nantes, July 2000.* pp. 44-50.

[3] Cardell-Oliver, Rachel. (2000). Conformance Testing of Real-Time Systems with Timed Automata. In *Formal Aspects of Computing.* (12):350-371

[4] Hennessy, M. C. (1988). *Algebraic Theory of Processes.* MIT Press.

[5] Henzinger, Th. A. (1996). The Theory of Hybrid Automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, pp. 278-292.

[6] Hoare, C.A.R. (1985). *Communicating sequential processes.* Prentice-Hall International, Englewood Cliffs NJ.

[7] Meyer, Oliver (2001). *Structural Decomposition of Timed CSP and its Application in Real-Time Testing.* Dissertation, Universität Bremen, FB 3.

[8] Nielsen, Brian (2000). *Specification and Test of Real-Time Systems.* Ph. D. Thesis, Department of Computer Science, The Faculty of Engineering and Science, Aalborg University, Publication No. 12.

[9] Peleska, Jan and Siegel, Michael. (1996). From Testing Theory to Test Driver Implementation. In M.-C. Gaudel and J. Woodcock (Eds.): FME '96: Industrial Benefit and Advances in Formal Methods. LNCS 1051, Springer-Verlag, Berlin Heidelberg New York 538-556.

[10] Peleska, Jan (1996). Test Automation for Safety-Critical Systems: Industrial Application and Future Developments. In M.-C. Gaudel and J. Woodcock (Eds.): FME '96: Industrial Benefit and Advances in Formal Methods. LNCS 1051, Springer-Verlag, Berlin Heidelberg New York pp. 39-59.

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

[12] Peleska, Jan and Siegel, Michael. (1997). Test Automation of Safety-Critical Reactive Systems. *South African Computer Jounal* 19:53-77.

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

[14] Peleska, Jan and Buth, Bettina. (1999). Formal Methods for the International Space Station ISS. In E.-R. Olderog, B. Steffen (Eds.): Correct System Design, Springer LNCS 1710, pp. 363-389.

[15] Peleska, Jan (2002). Hardware/Software Integration Testing for the new Airbus Aircraft Families. In *Proceedings of the TestCom 2002 Testing Internet Technologies and Services, Berlin, March, 19th-22nd 2002.*

[16] Peleska, Jan and Tsiolakis, Aliki (2002). Automated Integration Testing for Avionics Systems. In *Proceedings of the 3rd ICSTEST, International Conference on Software Testing, Düsseldorf, April 17th – 19th.*

[17] Peleska, Jan (2002). Formal Methods for Test Automation – Hard Real-Time Testing of Controllers for the Airbus Aircraft Family. Invited Talk. In Proceedings of the Sixth Biennial World Conference on Integrated Design & Process Technology (IDPT2002), Pasadena, California, June 23-28, 2002.

[18] Roscoe, A. W. (1998). *The Theory and Practice of Concurrency.* Prentice-Hall International, Englewood Cliffs NJ.

[19] Verified Systems International GmbH (1998). *RT-Tester Test Automation System.* Information available under `http://www.verified.de`.

[20] Sadeghipour, Sadegh (1998). *Testing Cyclic Software Components of Reactive Systems on the Basis of Formal Specifications.* Schriftenreihe Forschungsergebnisse zur Informatik, Bd. 40. Verlag Dr. Kovaĉ, Hamburg.

[21] Schneider, Steve (1995). An Operational Semantics for Timed CSP. *Information and Computation*, 116:193–213.

[22] Tretmanns, Jan (1992). *A Formal Approach to Conformace Testing.* PhD Thesis, University of Twente, Haag.