Automated Integration Testing for Avionics Systems

Peleska, Jan and Tsiolakis, Aliki
University of Bremen - TZI
Bremen
Germany
jp@tzi.de, tsio@tzi.de

In this presentation, the authors introduce a novel approach for integration testing which is currently applied in practice in the test of control systems for the Airbus aircraft family (A340-500/600, A318 and A380). This approach differs from conventional ones in two ways:

- It provides a unified concept and test automation technology from software integration testing to systems integration testing,
- it supports automatic test generation, execution and test evaluation based on real-time state machines operating in parallel.

To explain the first of these two characteristics in more detail, consider the conventional testing approach, where at least three different types of test specifications, test data and often different associated tools are applied:

- On module testing level, tests are performed against low-level software requirements (i.e., design specifications), and the test data refers to software objects (e.g., function interfaces, global variables).
- On software integration level, tests are performed against high-level software requirements, and the test data refers again to software objects.
- On HW/SW integration level, tests are performed against high-level software requirements, and the test data refers to HW interfaces of the controller under test.
- On systems integration level, tests are performed against system requirements, and the test data refers to HW interfaces, networks, and the interfaces of peripherals.

While many of the functional requirements to be tested are visible on both software and HW/SW integration level, the associated test specifications and test execution data cannot be easily compared, since the former refer to software objects, while the latter refer to hardware interfaces. Similarly, systems integration tests often investigate the same functional features as HW/SW integration tests, but use different interfaces (e.g. a real peripheral instead of data passed along a controller interface) to stimulate the responses of the system under test. Due to this terminology breach, results achieved on software integration test level were hard to compare to results on HW/SW integration or system level, and it was practically impossible to re-use (parts of) test specifications developed for different levels. Moreover, automated test data generation and test evaluation techniques could only be applied on a specific level, since the data representations were different. This often resulted in completely different tool sets to be applied on all testing levels.

To encounter this problem, our testing approach uses a concept called interface abstraction: Test specifications use abstract terms to describe inputs, outputs and other events (errors, warnings, requirements tracing information etc.) occurring during a test execution. These abstract terms consist of channel names associated with vectors of abstract data values. The channel names can be defined freely, for example using interface names introduced in the applicable specification documents. For concrete test executions, the abstract interface terms are mapped onto concrete interfaces of the system under test (SUT). This is performed by testsystem components called Interface Modules. For software integration testing, Interface Modules map abstract test data to concrete software interfaces such as

function calls with associated parameters, shared variables, buffers etc. Conversely, outputs of the SUT software are transformed back into the abstract representation. For systems integration testing, the Interface Modules are exchanged by a new set of modules accepting the same abstract terms, but transforming them to hardware interface driver calls instead of software interfaces.

The software of our test system which is responsible for automatic test generation, execution and evaluation operates in hard real-time on the level of abstract terms. As a consequence, a test specification which is meaningful for both software integration and systems integration can be re-used without modifications, just by exchanging the associated layer of Interface Modules. Interface abstraction has another advantage: It is not mandatory that abstract channels should be in one-to-one correspondence to concrete interfaces. Instead, whole sequences of concrete inputs and outputs may be abstracted to a single event of an abstract channel. This can be used when the SUT produces large numbers of interface data making test execution logs unreadable, or for specifying standard phases of the test execution with a single abstract term.

To explain the second characteristic of our approach, consider the way conventional systems integration testing was performed: Test cases consisted of a list of sequential steps, each step either describing a stimulus to be exercised on the system under test or a checking condition for the expected results, i.e. the SUT responses. The test cases were either executed manually or by means of computers acting as test drivers by interpreting test scripts with formal syntax in a sequential manner. This approach has two major draw backs: First, the effort for preparing tests is proportional to its duration - the longer the test execution should last, the larger the number of test steps to be defined in the test script. If test steps had to be written manually, it was completely infeasible to design test cases consisting of more than a few thousand steps at most. This approach is completely unacceptable for long-term testing, where the system under test should be stressed over several days or weeks, involving millions of I/Os. Second, the sequential nature of these test executions is not appropriate for the inherently parallel nature of the systems which are tested: In the real operational environment, the systems under consideration must be able to cope with many stimuli triggering different functions in parallel. Practical experience shows, that these situation also reveal the most critical errors, whereas many systems perform correctly as long only one function is exercised at a time.

To deal with this problem, our test system uses several so-called Abstract Machines operating in parallel, instead of a single sequential script interpreter: Each Abstract Machine interprets a state machine specification in hard real-time. Each state of the machine specifies the set of inputs to the SUT which are possible and the set of SUT outputs which comply to the expected SUT behaviour in the current state. Moreover, the state machine specifications encode all timing conditions when SUT reactions should occur or how long the testing environment should wait before generating a certain SUT input. Generation of an input to the SUT or reception of an SUT output leads to a state transition in the Abstract Machine. At test execution time, the Abstract Machine selects events from the set of possible inputs to the SUT and marks them as covered. When the same state is entered again, other possible inputs may be selected. This results in automatic generation of test input sequences to the SUT. When an SUT output is received by the Abstract Machine as an abstract event, it is checked whether the output fits into the set of events specified as legal in the current state of the Abstract Machine. The checks can be performed with respect to correctness of the data, the sequencing and the timing of SUT outputs. This solves the task of automatic evaluation of SUT reactions. In a test execution, an arbitrary number of Abstract Machines may cooperate in parallel, thereby simulating different parallel components in the operational environment of the SUT and checking different aspects of the SUT behaviour.

The concepts described in this presentation are illustrated by a (simplified) example concerning integration testing of the smoke controller component for the Airbus A318 aircraft. These tests have been performed by the authors' research team in cooperation with Verified Systems International GmbH for KID-Systeme, the company responsible for the development of the new smoke controller. A part of the results described in the presentation has been elaborated within the EU project VICTORIA which focuses on research for a new generation of aircraft electronics and associated verification, validation and testing methodology.