# Testing Operating Systems with RT-Tester

Jan Peleska,
Oliver Meyer, Johannes Kanefendt and Florian Lapschies
jp@verified.de

University of Bremen and Verified Systems International GmbH, Bremen, Germany

Airbus – 2007-01-19
2006-11-06

# Overview

Objectives: Perform extensive tests for avionics operating system – fulfil requirements of RTCA DO-178B for level A applications

- Tests in simulation environment versus on-target testing
- Generic test configuration for operating systems
- RT-Tester test automation system
- Example: ARINC 653 operating system test in Linux simulation environment with RT-Tester

# Tests in simulation environment versus on-target testing

- On-target tests are required for
  - Proving functional correctness of HW/SW integration (c. f. IMA Bare Module Tests)
  - Achieving structural coverage on target HW (certification requirement)
- On-target tests for embedded systems usually complicate white-box testing
- On-target tests require separation of (parts of) test equipment and SUT

# Tests in simulation environment versus on-target testing

Simulation environments (PC, work station)

- facilitate white-box testing
- may not be used for certification credit if simulation architecture differs “too much” from target architecture
- require special care for operating system tests:
  - For functional integration testing, conflicts between tested OS System Under Test OS(SUT-OS) and simulation platform OS have to be avoided – for example: Scheduler, virtual memory management and partitioning, interrupt relaying
  - Access of SUT-OS to target HW has to be stubbed – in some cases by access functions to simulation platform HW

# Generic test configuration for operating systems

For functional testing:

- The non-existing application layer is replaced by test agents (= test applications) that exercise the application layer interface (API) (= APEX) of the SUT-OS
- Test agents
  - are re-usable
  - can be remotely controlled by testing environment
  - can exercise the most general behaviour at the SUT-OS API which is possible for “real” applications
  - may possess pre-programmed scenarios for robustness testing and time-critical API call sequences

# Generic test configuration for operating systems

- Testing environment exercises calls to SUT-OS via remote method invocation to test agents
- SUT-OS is extended by internal test functions for
  - Checking internal data structures
  - Tracing internal behaviour of SUT-OS kernel functions
  - Tracing and storing code coverage information
- Internal test functions can be triggered by test agents via auxiliary API calls that also recover test results

# Generic test configuration for operating systems

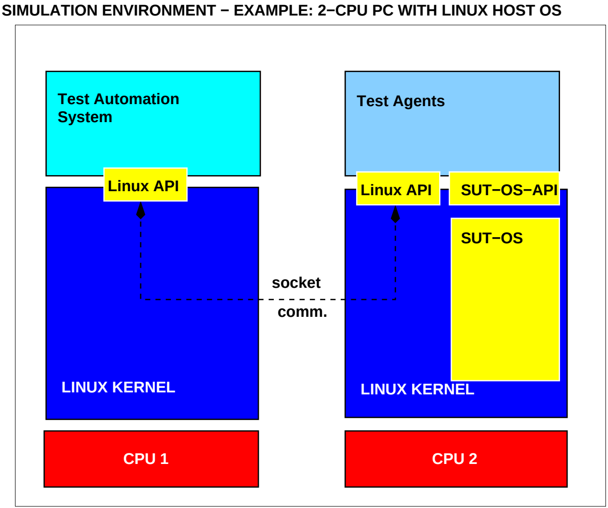
Framework for embedding SUT-OS into simulation platform: This has been performed by University of Bremen for ARINC 653 operating system implementation on Linux

- On simulation system boot, memory is exclusively reserved for SUT-OS:
  - pages locked in memory
  - separate memory map managed by SUT-OS kernel
- Simulation platform clock interrupt triggers Linux scheduler and SUT-OS scheduler in alternation
- Configured SUT-OS processes (=partitions on ARINC 653 OS) are initialised by SUT-OS init process

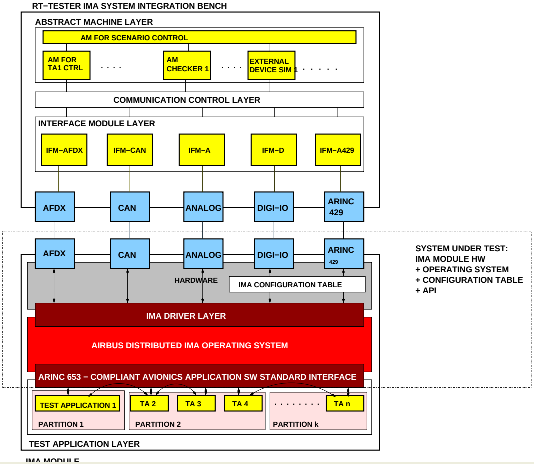
# Generic test configuration for operating systems

- On ARINC 653 OS, every partition runs one or more test agents (= threads of the partition)
- SUT-OS API triggers different trap (software interrupt) which leads to SUT-OS kernel instead of Linux kernel
- Test agents may combine Linux and SUT-OS API calls since these are distinguished by different traps
- Communication with testing environment is performed via TCP/IP socket communication (Linux API)

# Generic test configuration for operating systems



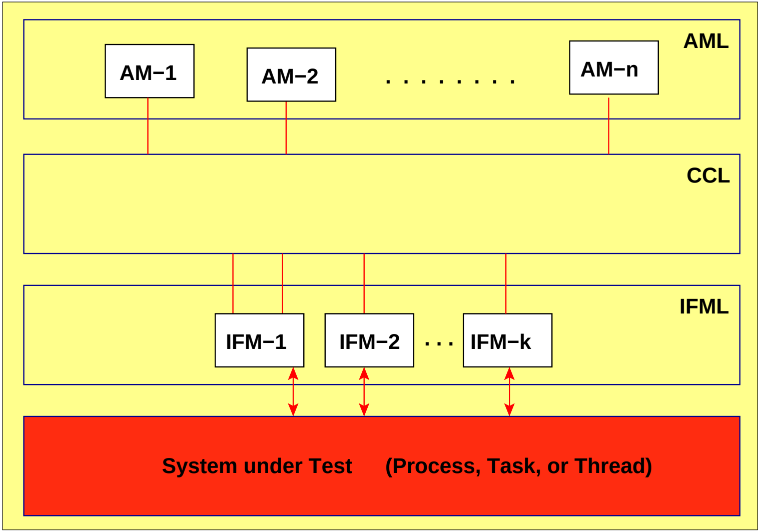
# Example for On-target testing: Bare Module Tests / Configured Module Tests of IMA ARINC 653 operating system



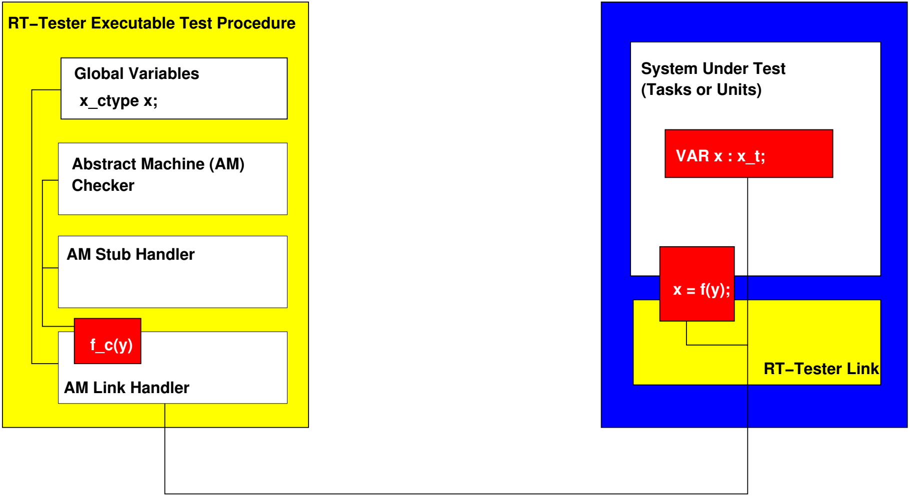
# RT-Tester test automation system

- Distributed simulation and test system
- Abstract machines/Interface modules run in parallel to perform
  - Simulations
  - Automated on-the-fly checking
  - Stimulation of SUT
- Interface abstraction by means of channels and vectors
- Hard real-time capabilities
- Single-CPU, Multi-CPU and cluster hardware configurations available
- Supports all testing phases – from unit tests to system integration tests
- Provides powerful test automation mechanisms

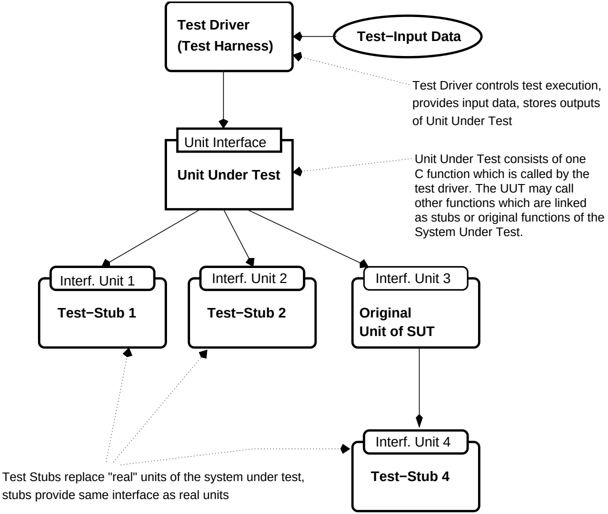
# Software Integration Test with RT-Tester



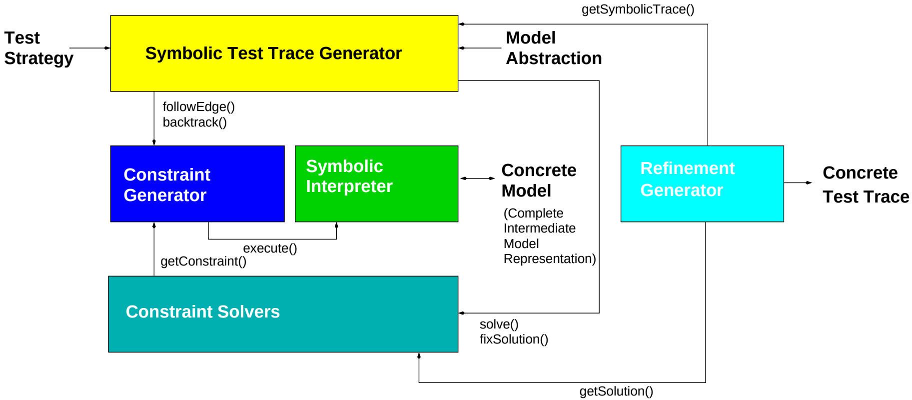
# Remote Method Invocation For Test Control



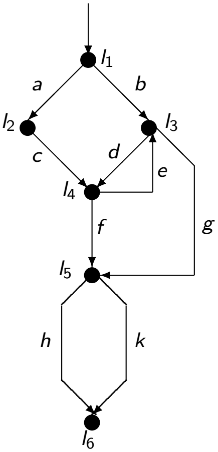
# Module Test Configuration



# Test Cases, Test Data Generation Framework



# SUT: Time-Discrete Input-Output Hybrid Systems




Initial conditions

$Init(l_1) = \texttt{true}$,

$\forall l \in Loc - \{l_1\} : Init(l) = \texttt{false}$

Transition labels $a, \ldots, k$:

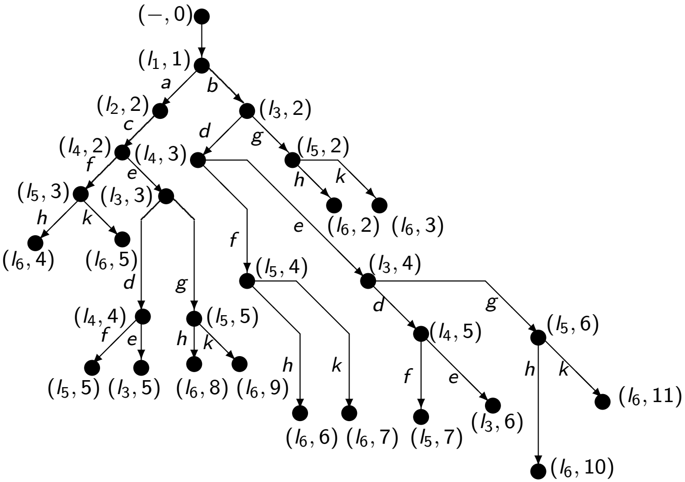
$a = (l_1, \texttt{true},$
$\quad ((x_1, x_2), (in_1, in_2)), l_2)$

...

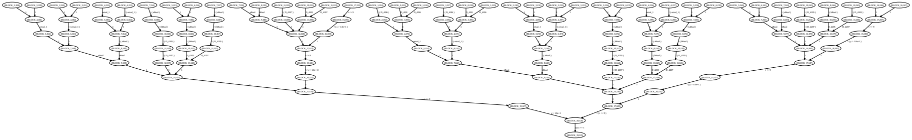
$d = (l_3, x_1 < \exp(x_1 \cdot x_2),$
$\quad ((x_1, x_2), (x_1^2 \cdot \sin(x_2), x_2)), l_4)$

...

# Symbolic Test Case Tree STCT



# Reachability Trees for MCDC Coverage



# Example: ARINC 653 operating system test in Linux simulation environment with RT-Tester

Test examples for

- ARINC 653 semaphore mechanism
- ARINC 653 memory management
- Code coverage analysis
- Test case coverage