Hard Real-Time Test Tools – Concepts and Implementation

Prof. Dr. Jan Peleska
Centre for Computing Technologies, University of Bremen, Germany

Dr. Ing. Cornelia Zahlten
Verified Systems International GmbH, Bremen, Germany

4th ICSTEST International Conference on Software Testing
April 2- 4, 2003, Cologne

In this presentation, ...

- ... we describe concepts and techniques for automated testing of hard real-time systems
  - Test specification formalisms describing rules for automated
    - Discrete and time-continuous test data generation
    - Test evaluation (“test oracles”)
  - Hardware and operating system support for testing in hard real-time

Background and Related Work

- Theoretical foundations of the modelling techniques used have been elaborated by
    - T. A. Henzinger (Hybrid Automata)
    - Authors' research teams at TZI and Verified Systems (algorithms for automatic test data generation and test evaluation)
    - Brinksma, Cardell-Oliver, Tretmans, Nielsen et. al. (alternative approaches to test automation)
    - E. Bryant (ordered binary decision diagrams)

- Real-time concepts are based on / inspired by results of
    - T. A. Henzinger (GIOTTO real-time programming language)
    - H. Kopetz (Time-Triggered Architecture for real-time systems)
    - Authors' research teams at TZI and Verified Systems (Linux real-time kernel extensions, user thread scheduling, unified communication concept)
    - ARINC 653 Standard for avionics operating system API

Background and Related Work

- All concepts described here have been implemented in Verified's test automation tool RT-Tester
- Applications are currently performed for SW integration testing – HW/SW integration testing – system testing of
  - Aircraft controllers for the Airbus families:
    - A318-SDF Smoke Detection Facility
    - A318/A340-500/600 CIDS Cabin Communication System
    - A380 IMA Modules – controllers with Integrated Modular Avionics architecture
  - Train control and interlocking components (Siemens)
- RT-Tester automation tool has been qualified for testing specific A/C controllers according to RTCA DO-178B

Recall: Hard Real-Time Testing ...

- ... Investigates the behaviour of the system under test (SUT) with respect to correctness of
  - Discrete data transformations
  - Evolution of continuous observables over time – speed, temperature, thrust, …
  - Sequencing of inputs and outputs
  - Synchronisation
  - Timing of SUT outputs with respect to deadlines – earliest/latest points in time for expected outputs

A Glimpse at Theory: Test Specification Formalisms for Hard Real-Time Systems

- Question: How much expressive power is required for “suitable” hard real-time systems test specification formalisms?
- Answer from theoretical research (Hybrid Automata): Formalisms need to express facts about
  - States and events
  - Cooperating parallel system components
  - Initial conditions – invariants – flow conditions
  - Trigger conditions for state transitions
  - Actions

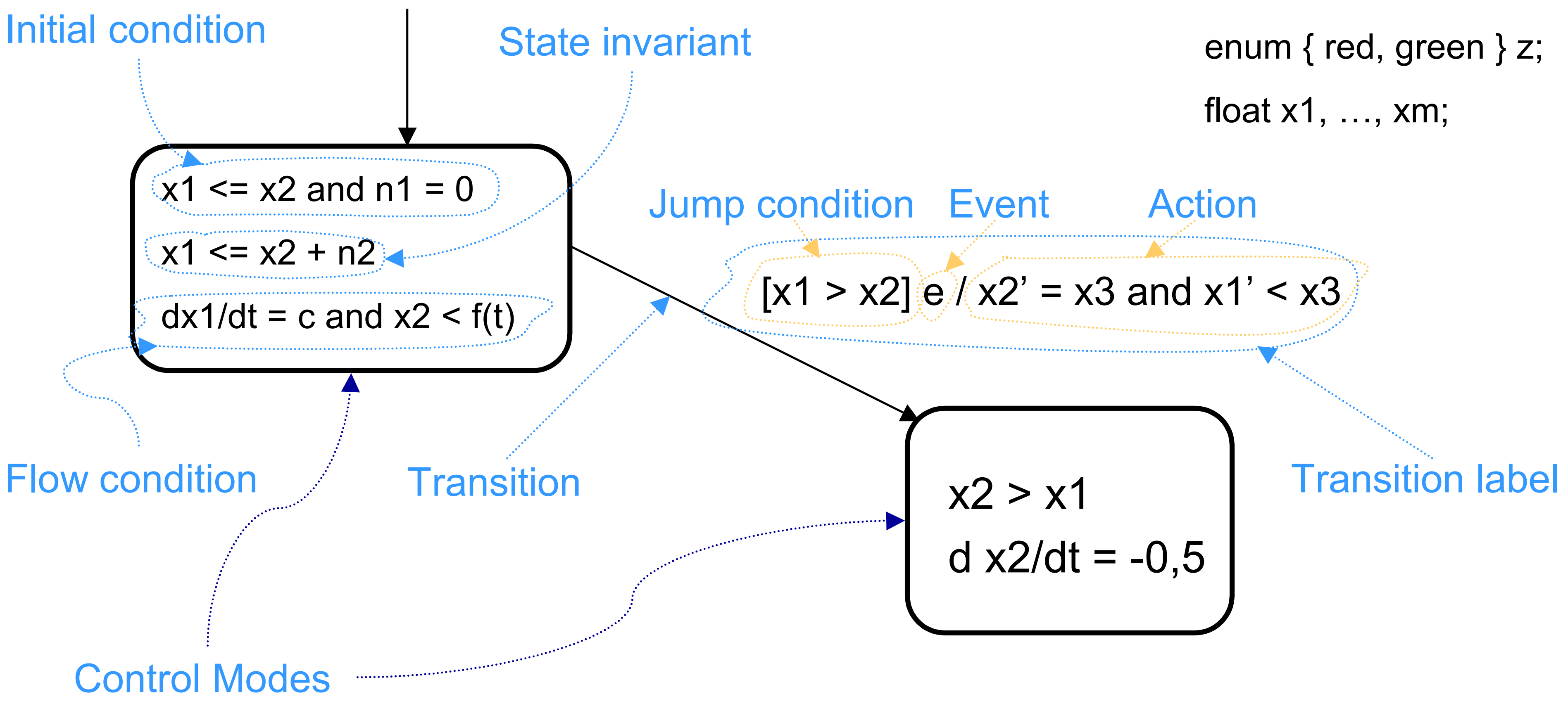
Hybrid Automaton (one sequential component)

State Variables

int n1, … , nk;

enum { red, green } z;

float x1, …, xm;



Initial condition

State invariant

x1 <= x2 and n1 = 0

x1 <= x2 + n2

dx1/dt = c and x2 < f(t)

Jump condition Event Action

[x1 > x2] e / x2’ = x3 and x1’ < x3

Flow condition

Transition

Transition label

x2 > x1

d x2/dt = -0,5

Control Modes

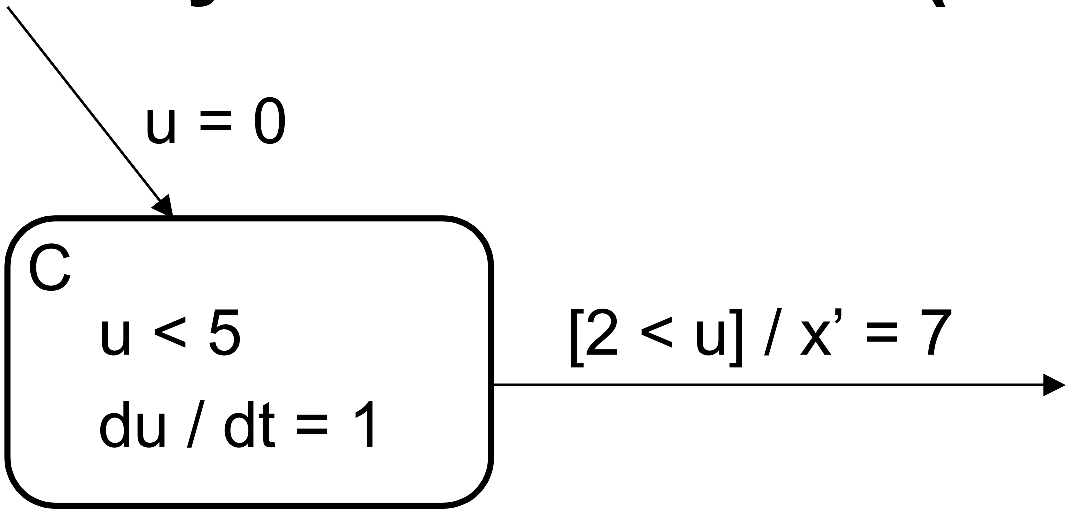
Hybrid Automata

- Control Modes: Principal states describing the operational modes of the (sub-)system
- State Variables: discrete variables (int, enum, …) and continuous variables (float, complex,…)
- State Space = control modes + state variables
- Transitions: change between control modes
- Labels: transition specification
    - Jump condition: must hold for variables
    - Event: input signal which triggers transition if jump condition holds
    - Action: list of output signals and predicate specifying how variables are changed when transition occurs – may be deterministic (x’ = 5) or nondeterministic (x’ < y)

Hybrid Automata (continued)

- Control modes and variables may be changed when transitions take place
- Continuous variables change over time according to the flow condition specified for actual control mode
- System may stay in control mode as long as the associated state invariant holds
- System may take transition as soon as jump condition holds and (optional) input event occurs
- This concept allows to specify deadlines for system reactions via invariants and jump conditions

Hybrid Automata (continued)



After entering control mode C, system will leave this mode within time interval (deadline) [2,5) time units, setting x to 7.

Adapting Theory to Real-Time Testing Practice: A List of Problems

For practical hard real-time testing, the following problems have to be solved:

- Interface abstraction:
  - How should SUT interface data be abstracted in test specifications ?
  - How is SUT interface data mapped to abstract specification data and vice versa ?
- Communication concept:
  - How should parallel test system components interact with each other and with SUT ?

Adapting Theory to Real-Time Testing Practice: A List of Problems

- Parallel execution: How can
  - Stimulation of test-specific SUT reactions
  - Simulation of environment components
  - Checking of SUT reactions

  be performed in parallel and in real-time ?
- Generation of input data: How should SUT input ports be stimulated in real-time, in order to
  - Trigger specific SUT reactions (transitions)
  - Establish invariant conditions in specific SUT states
  - Establish flow conditions on continuous SUT inputs ?

Adapting Theory to Real-Time Testing Practice: A List of Problems

- Checking of output data: How can we check SUT outputs against
  - State transitions describing the expected SUT behaviour
  - State invariants and
  - Flow conditions which should be enforced by SUT

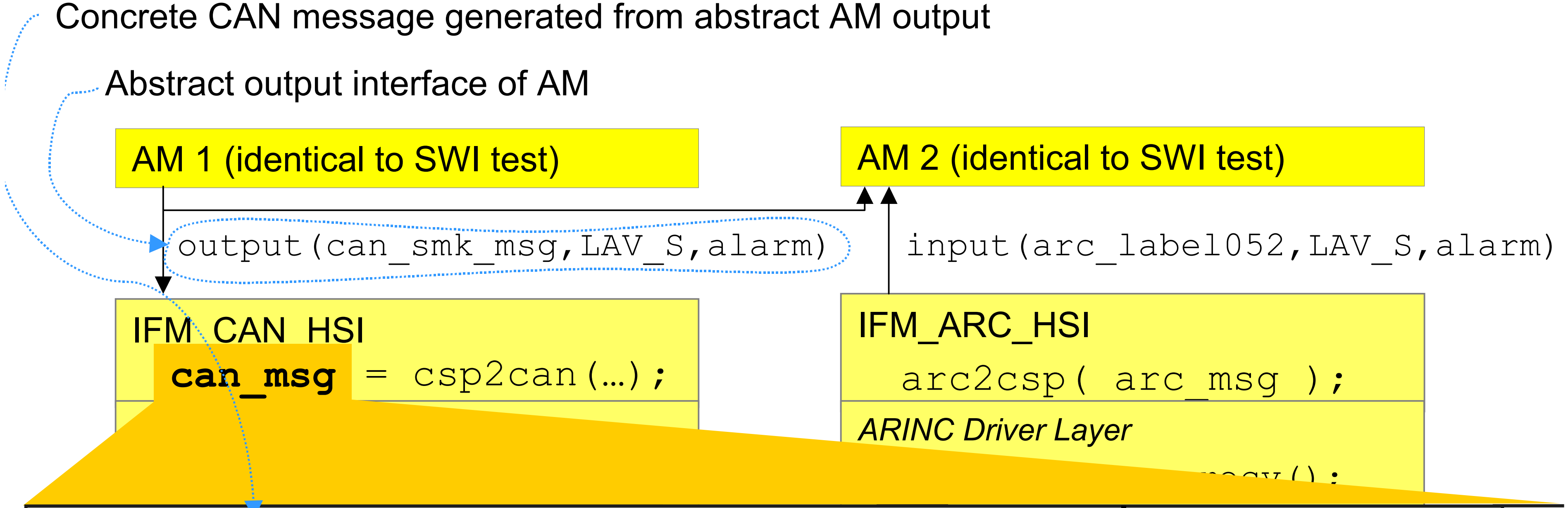
- preferably on-the-fly ?

Adapting Theory to RT-Testing Practice: Solutions

Interface Abstraction

- Interface Modules are used as adapters between test specifications and SUT interfaces (SW or HW interfaces)
- Events and state variables are refined to the concrete SUT input interfaces and associated data
- SUT outputs are abstracted to the events and variable values used on test specification level.

Example: Interface Abstraction



Concrete CAN message generated from abstract AM output

Abstract output interface of AM

AM 1 (identical to SWI test)

AM 2 (identical to SWI test)

`output(can_smk_msg,LAV_S,alarm)`

`input(arc_label052,LAV_S,alarm)`

IFM_CAN_HSI

`can_msg = csp2can(…);`

IFM_ARC_HSI

`arc2csp( arc_msg );`

ARINC Driver Layer

CAN Message Identifier																						CAN Data Frame							
28	27	26	25	24	…	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
							0	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
Msg Type				Fct Code			Module ID **„LAV_S“**										System Id **„Smk Detection System“**					Byte 1 **„Alarm“**							

Adapting Theory to RT-Testing Practice: Solutions

- Communication concept
  - On abstract level, all interfaces are identified as ports
    - Sampling ports offer operations
      - Read and keep current data value in port
      - Write new value to port

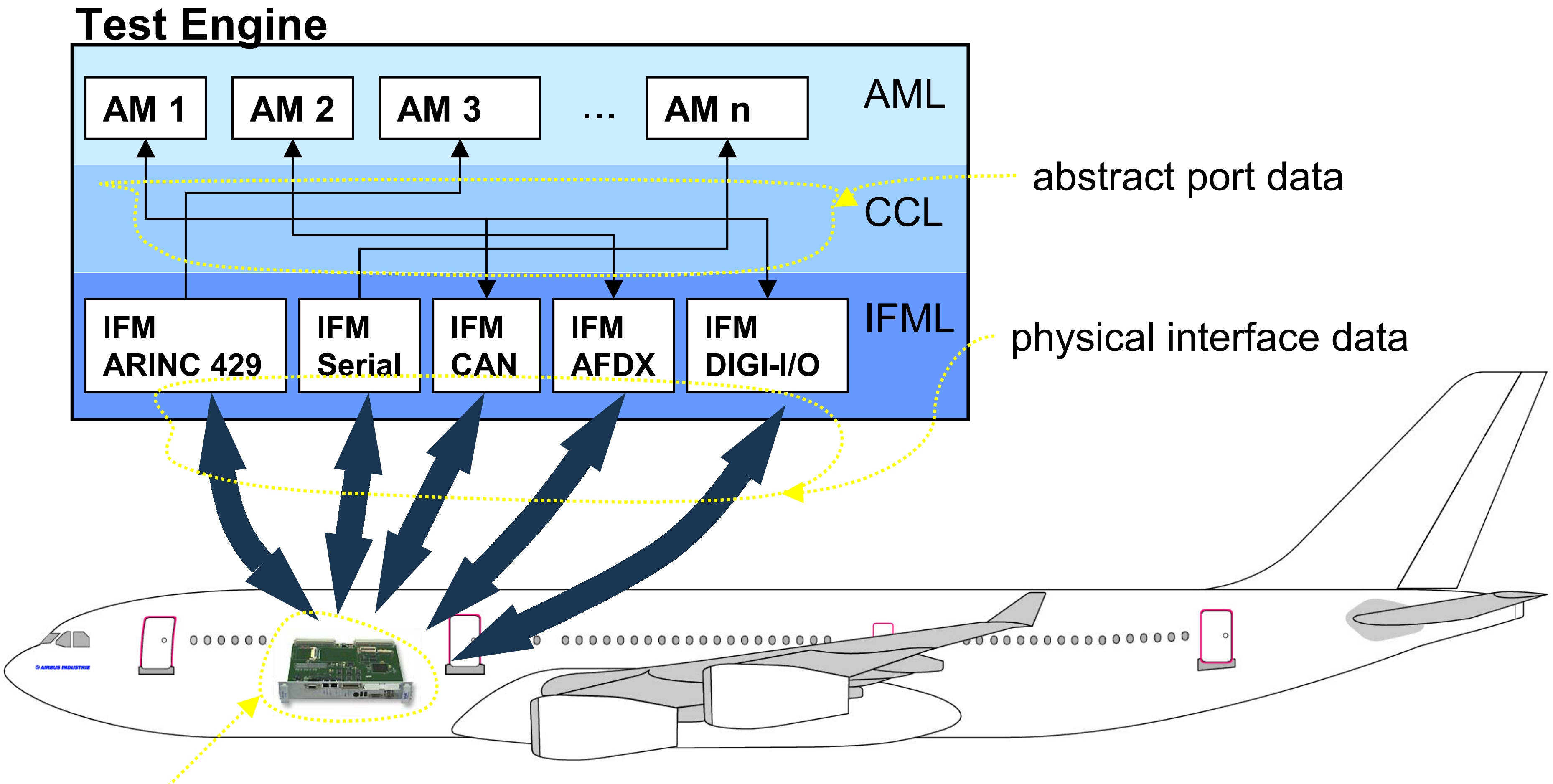
      Used for communication of sensor/actuator data and state variables
    - Queuing ports are FIFO buffers with operations
      - Append to end of queue
      - Read and delete first element of queue
      - Read and keep first element of queue

      Used for communication of messages and events

Adapting Theory to RT-Testing Practice: Solutions

- Parallel execution:
  - Parallel components are allocated as Abstract Machines on dedicated Light Weight Processes (LWPs)
  - Light weight processes in multi-processor environments may use CPUs exclusively
  - User thread scheduling of Abstract Machines on LWPs without participation of the operating system kernel
  - Port communication mechanism is implemented by Communication Control Layer

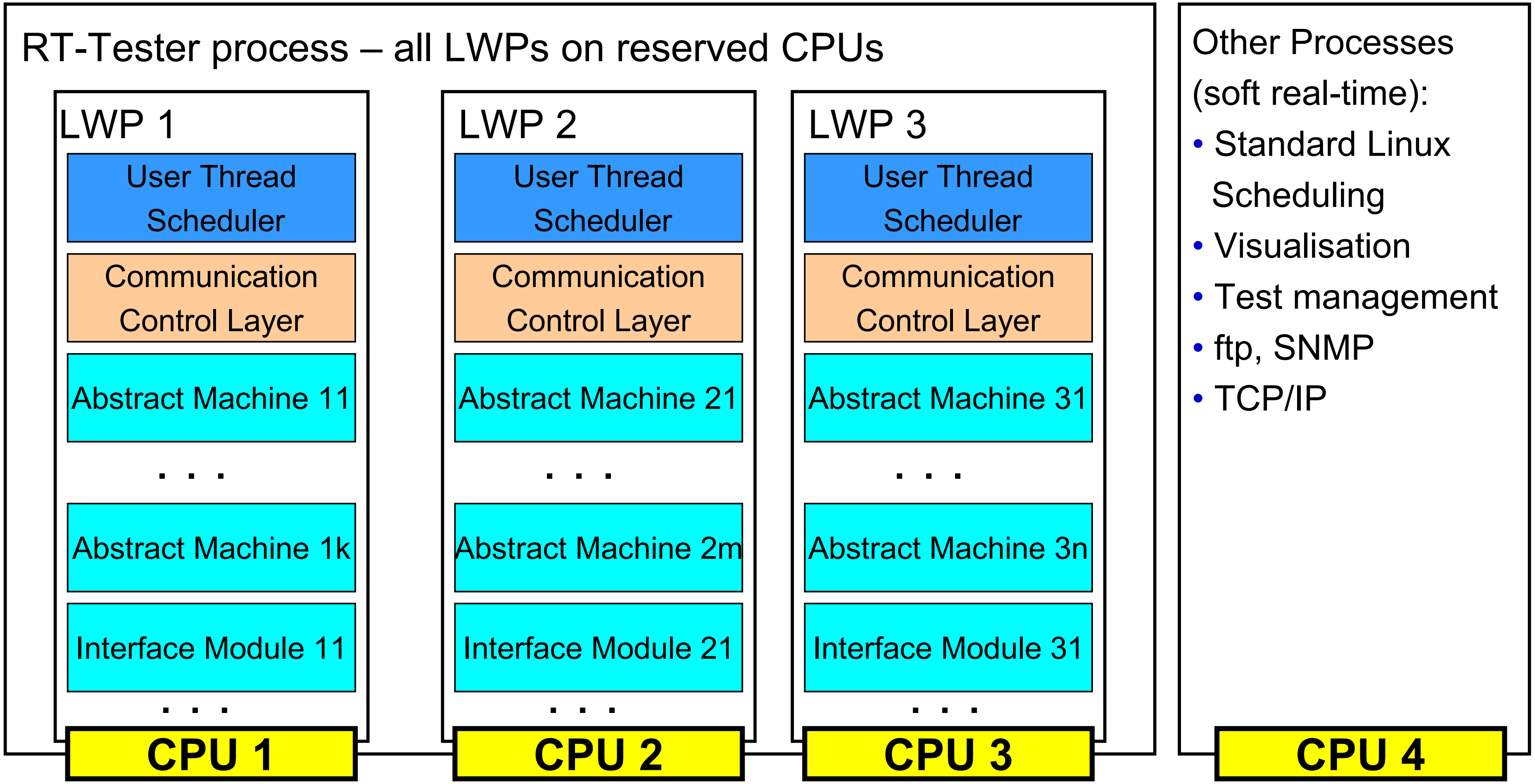
RT-Tester Organisational Model for Testing A/C Controllers



System Under Test : A318-SDF, A318/A340 CIDS, A380 IMA Module

Solutions … LWPs, Abstract Machines and Interface Modules

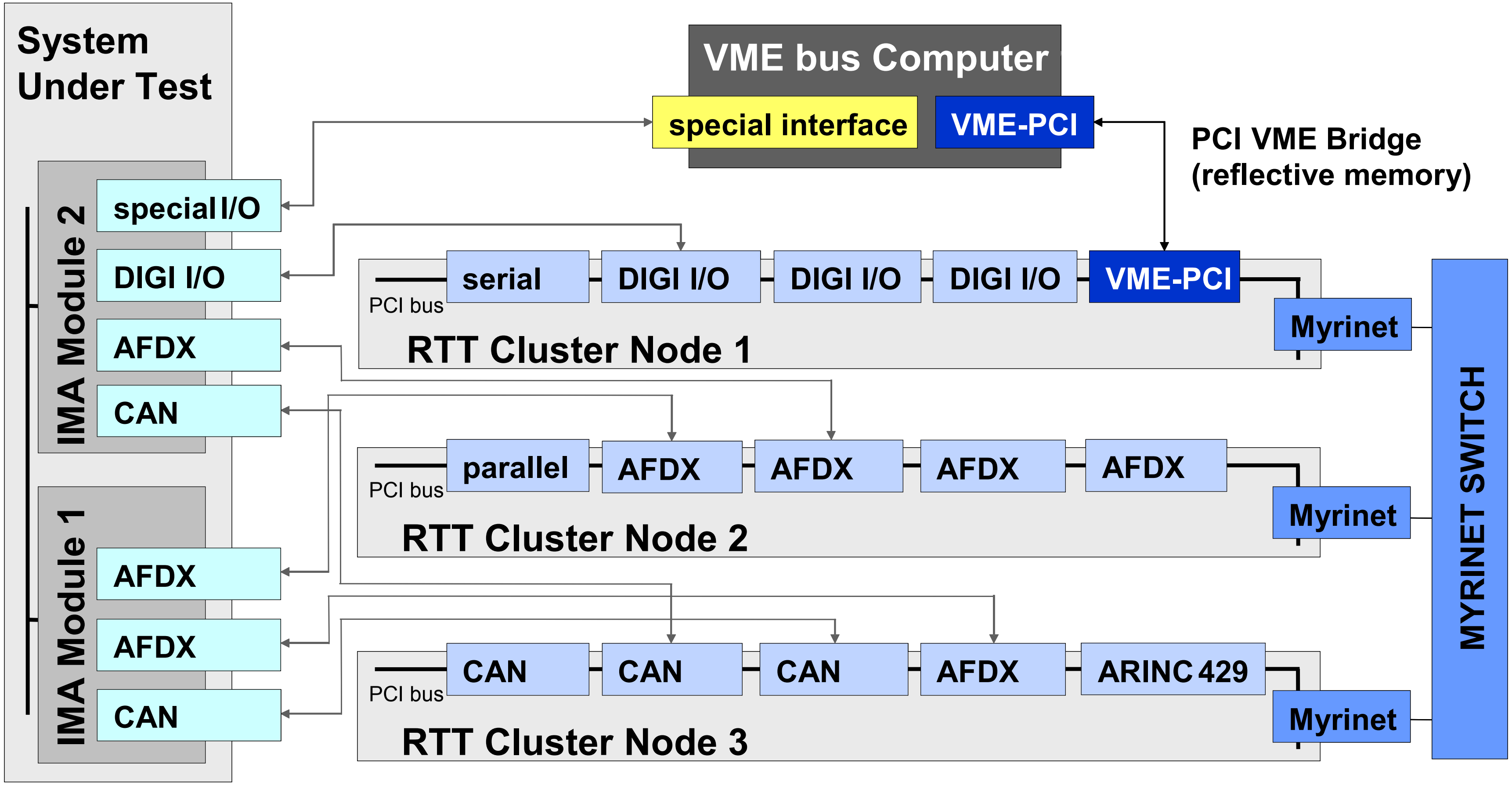
RT-Tester Engine



Adapting Theory to RT-Testing Practice: Solutions

- Parallel execution (continued):
  - Explicit mapping from I/O interrupts to CPUs
  - High resolution real-time clock and timers
  - Avoid PCI bus and memory bus bottlenecks by means of test engine cluster consisting of 2 or more PCs
  - Communication between cluster nodes via high-speed message passing (DMA) over Myrinet link
  - Accuracy better than 100microsec without using specialised hardware

Test Engine Cluster Configuration for A380 IMA Testing



System Under Test

IMA Module 2: special I/O, DIGI I/O, AFDX, CAN

IMA Module 1: AFDX, AFDX, CAN

VME bus Computer: special interface, VME-PCI

PCI VME Bridge
(reflective memory)

RTT Cluster Node 1: PCI bus, serial, DIGI I/O, DIGI I/O, DIGI I/O, VME-PCI, Myrinet

RTT Cluster Node 2: PCI bus, parallel, AFDX, AFDX, AFDX, AFDX, Myrinet

RTT Cluster Node 3: PCI bus, CAN, CAN, CAN, AFDX, ARINC 429, Myrinet

MYRINET SWITCH

Adapting Theory to RT-Testing Practice: Solutions

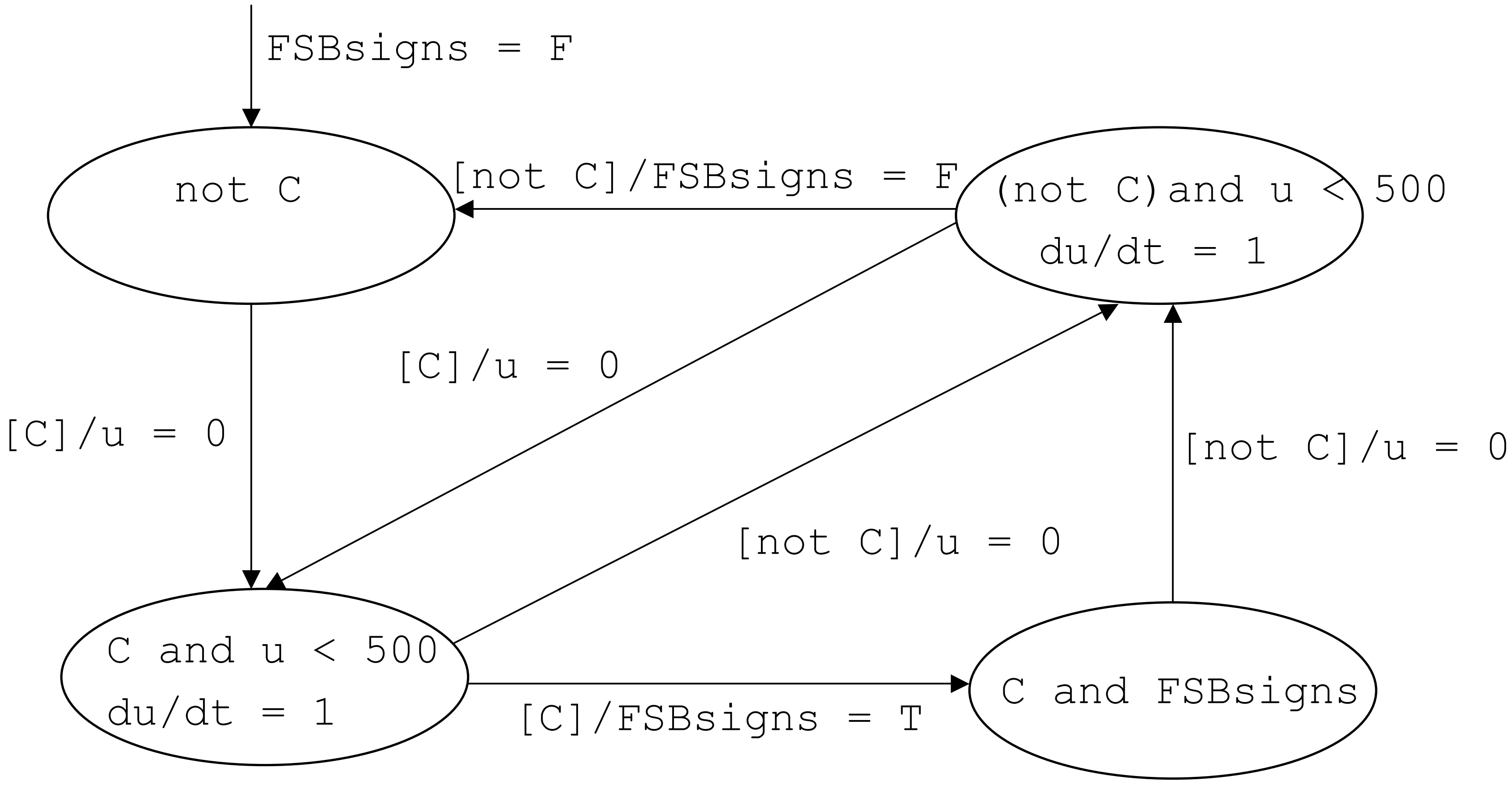
- Generation of input data – example: Control of Fasten Seat Belts Signs – switch FSB signs on (`FSBsigns = true`) within 500msec if
  - Cockpit switch `FSBswOn` has been activated or
  - Cabin pressure is low (`CPC1on or CPC2on`) and automatic FSB switching has been configured (`CONF_FSB_CPC`)for this situation
  - Landing gears are down and locked (`LDGdownLck`) and automatic FSB switching has been configured (`CONF_FSB_LDG`)

Adapting Theory to RT-Testing Practice: Solutions

- Generation of input data – example:
  Logical condition C for “FSB SIGNS ON”:
  - C ≡

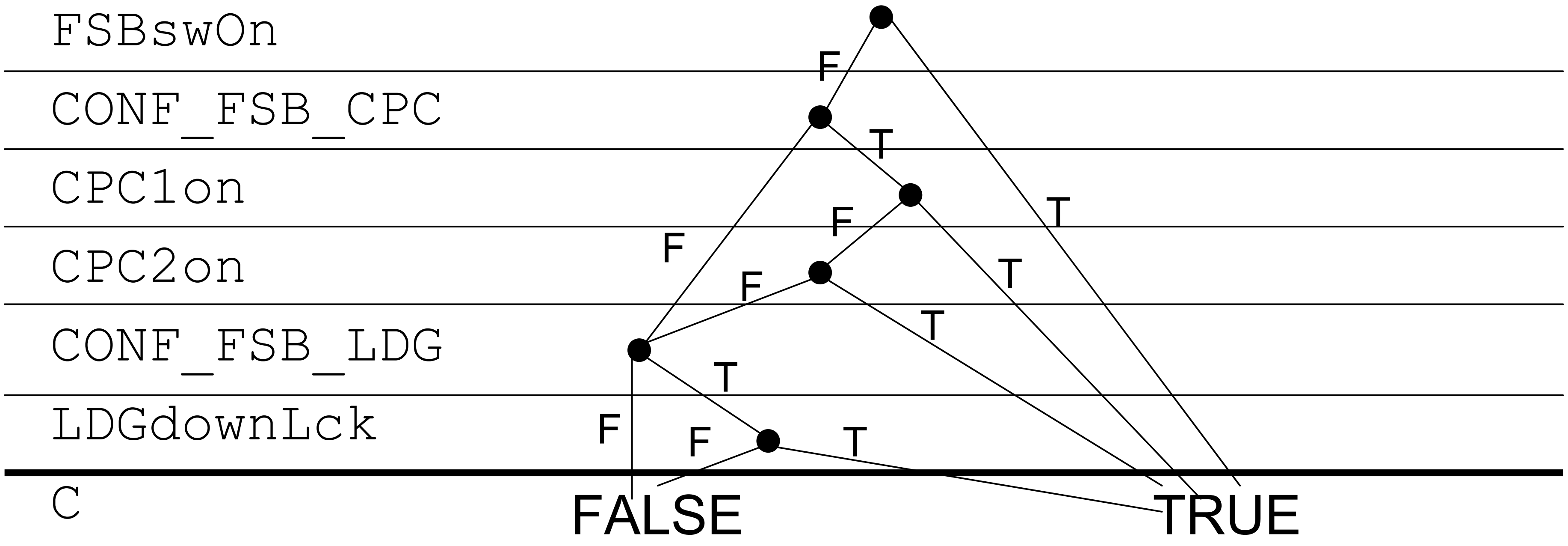
```
 FSBswOn or
  (CONF_FSB_CPC and (CPC1on or
CPC2on))or
  (CONF_FSB_LDG and LDGdownLck)
```

Example – continued: specification of FSB controller



Example – continued: Input Generation for Condition C

- Automatic generation with Ordered Binary Decision Diagrams (OBDD): Every path of OBDD defines combination of input values to make C true or false



Example – continued: Input Generation for Condition C

- Test system simulates SUT transitions between control modes in parallel to SUT execution
- In each control mode, test system generates input data vector, so that
  - Every possible transition will be taken
  - Every possible data combination for making conditions true or false is generated from OBDD
  - If too many combinations exist, heuristics are applied to generate “relevant” combinations – users may specify such combinations to optimise data generation process

Conclusion

- Hybrid Automata have suitable expressive power for testing real-time systems with both discrete and time-continuous interfaces (sensors, actuators)
- For using Hybrid Automata in the context of testing,
  - A hard real-time testing environment has been developed based on
    - Port communication
    - Network of cooperating Abstract Machines (AM) performing test control, simulation and checking and
    - Interface Modules (IFM) for mapping data between AM and SUT interfaces
    - Specialised user thread scheduling for AM and IFM on reserved CPUs – hard real-time extension of Linux kernel
    - Test engine cluster platform based on multi processor PC linked via Myrinet

Conclusion

- For using Hybrid Automata in the context of testing (continued),
  - Test data generation algorithms have been developed based on
    - Graph traversal in real-time for coverage of control modes
    - User-specified selection of discrete input data to SUT or
    - Automatic selection of input data based on binary decision diagrams
    - Stepwise Δt-integration of flow conditions – solutions of differential equation may be imported from Matlab or similar tools

Conclusion

- For using Hybrid Automata in the context of testing (continued),
  - Algorithms for automatic evaluation of SUT responses ("Test Oracles") have been developed based on
    - Graph traversal algorithms for checking SUT outputs against expected transitions between control modes
    - Pre-compiled correctness conditions for checking invariants and jump conditions
    - Comparison of time-continuous SUT outputs on actuator interfaces against reference functions derived from flow conditions