Automated Integration Testing for Avionics Systems
Prof. Dr. Jan Peleska, Aliki Tsiolakisa
Center for Computing Technologies, Safe Systems
University of Bremen, Germany

3rd ICSTEST International Conference on Software Testing
18. April 2002
In this presentation, ...

- we describe a novel approach for integration testing, providing
  - A unified concept and test automation technology for all test levels --- from software integration testing to system integration testing
  - Automatic test generation, execution and test evaluation based on real-time state machines operating in parallel
Objectives of the Approach

- Provide a unified concept for re-usable test specifications
- Support automatic test generation, test execution and test evaluation
- Reduce the effort for test preparation
- Support modelling of
  - Environment simulators,
  - Test evaluation components

as parallel entities, following the architecture of the operational environment and of the system under test.
Conventional Testing Approach

<table>
<thead>
<tr>
<th>SUT</th>
<th>Test Tool</th>
<th>Test Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Module Testing</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f() {…}</td>
<td>Tool 1</td>
<td>low-level SW requirements</td>
</tr>
<tr>
<td>g() {…}</td>
<td>Tool 1</td>
<td></td>
</tr>
<tr>
<td><strong>Software Integration</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f() {...}</td>
<td>Tool 2</td>
<td>high-level SW requirements</td>
</tr>
<tr>
<td>g() {...}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gCtr</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HW/SW Integration</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW</td>
<td>Tool 3</td>
<td>high-level SW requirements</td>
</tr>
<tr>
<td><strong>System Integration</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEV1, DEV2</td>
<td>Tool 4</td>
<td>system requirements</td>
</tr>
</tbody>
</table>
Example

- Aircraft Smoke Detection Controller (simplified for illustration purposes)
  - Smoke Detectors are located in different areas of the aircraft (e.g., lavatories, cargo compartments)
  - Smoke Detectors send status messages to controller using the CAN bus
  - In case of a smoke alarm signalled by detectors, controller shall
    - Turn on *Smoke Warning Light* in cockpit
    - Indicate the alarm on the *Flight Warning System* by sending a message using the ARINC 429 bus

- Testing levels considered in this presentation: Software Integration, HW/SW Integration
Example: Conventional Testing – Software Integration (1)

SUT (Software Thread)

```c
void smkCtrl () {
    while (true) {
        msg = getSmkMsg();
        switch (msg.msgType) {
            case alarm:
                setSmkWarnLight(on);
                putArcMsg(msg);
                break;
            case ok:
                ...
            ...
        }
    }
}
```

Test Environment

```c
int main() {
    pthread_create(..., smkCtrl...);
    // Test case 1
    gCanMsg.loc = LAV_S;
    gCanMsg.msgType = alarm;
    wait(t); //SUT processes data
    if(gcArcMsg.loc!=LAV_S || gArcMsg.msgType!=alarm)
        print(``ERROR Test 1´´);
    // Test case 2...
}
```
Example: Conventional Testing – Software Integration (2)

Test Specification consists of

- Test Data: Assigned to global variable `gCanMsg` with type
  
  ```
  struct { location_t loc; msg_type_t msgType;
  }
  ```

- Checking Condition for Expected Results: Evaluation of global variable `gArcMsg` with same type.

- Test Stubs:
  - `getSmkMsg()`: return the value of the global variable `gCanMsg` defined by test environment to SUT which is SW thread `smkCtrl()`
  - `putArcMsg()`: Copy parameter value `msg` supplied by SUT to global variable `gArcMsg`, to be evaluated by test environment
Example: Conventional Testing – HW / SW Integration(1)

// Test Case 1
unsigned int can_msg[2] = {0,0};
can_msg[0] |= (3 <<25);       // Msg Type
can_msg[0] |= (0x01EF) <<15;  // Function Code
can_msg[0] |= (2 <<12);       // Module ID „smoke detector“
can_msg[0] |= ....
can_send(can_msg);...

wait(t); // SUT processes data
arc_msg = arc_recv ();
if (!(arc_msg & (1 <<10)) ) print(„ERROR Test 1“);
Example: Conventional Testing – HW/SW Integration (2)

Test Specification

- **Test Data:** CAN messages
  - Construction of the CAN message for each test case

- Checking Conditions for Expected Results: Evaluation of ARINC 429 messages
  - Checking the correctness of bit patterns in the ARINC message

Transmission of the test data using the CAN bus / ARINC 429 bus

- *Driver Layer* between the hardware and the software providing functions for sending and receiving the messages
## CAN and ARINC Messages

### CAN Message Identifier

<table>
<thead>
<tr>
<th>CAN Message Identifier</th>
<th>CAN Data Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>28 27 26 25 24 ... 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>0 1 0 1 0 0 0 0 0 0 0 0 0 1 1</td>
<td>0 0 0 0 0 0 1 0</td>
</tr>
</tbody>
</table>

### Module ID and System Id

- **Module ID**: „LAV_S“
- **System Id**: „Smk Detection System“

### CAN Message Identifier and System Id

- **Msg Type**: “Alarm”

### ARINC 429 Data Outputs Label 052

<table>
<thead>
<tr>
<th>ARINC 429 Data Outputs Label 052</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
</tbody>
</table>

### PARITY

- **CABIN**: Label 052
Disadvantages of the Conventional Testing Approach

- Different types of test specifications based on
  - Low-level software requirements
  - High-level software requirements
  - System requirements
- Different types of test data referring to
  - Software objects
  - Hardware interfaces
  - Hardware interfaces, networks and interfaces of peripherals
- Application of different associated tools

„incomparable“ test results, no re-use of test specs
Unified Concept and Test Automation Technology

- **Interface abstraction**
  - Test specifications use abstract terms for
    - inputs
    - outputs
    - errors, warnings, requirement tracing information, etc.

- Abstract Machines (AM) interpret the test specifications in real-time
  - Channels with associated vectors of abstract data values
    ```
    channel can_smk_msg:
      location.status
    channel arc_smk_msg:
      location.status
    ```

- Test execution:
  Mapping onto concrete interfaces of the SUT

- Interface Modules (IFM)
Example: Software Integration
Test using Interface Abstraction

AM 1 (Test data generator)
- `can_smk_msg!LAV_S.alarm`

AM 2 (Test checker)
- `arc_label052!LAV_S.alarm`

IFM_CAN_SWI
- SHM-CAN
  - `LAV_S alarm`

IFM_ARC_SWI
- SHM-ARC
  - `LAV_S alarm`

Test Stub
- `smkMsg_t getSmkMsg(){return}
  putArcMsg(...){...buffer[...]=msg...}`

SUT
- `void smkCtrl()
  msg=getSmkMsg();...
  putArcMsg(msg);...`
Example: HW/SW Integration Test using Interface Abstraction (1)

AM 1 (identical to SWI test)

```
IFM_CAN_HSI
    can_msg = csp2can(...);
    can_send(can_msg);
```

AM 2 (identical to SWI test)

```
IFM_ARC_HSI
    arc2csp(arc_msg);
    arc_msg = arc_recv();
```

CAN Bus

```
void smkCtrl() {
    msg=getSmkMsg();
    putArcMsg(msg);
}
```

ARINC Bus
Example: HW/SW Integration using Interface Abstraction (2)

AM 1 (identical to SWI test)

\[
\text{can\_smk\_msg!LAV\_S\_alarm}
\]

IFM CAN HSI

\[
\text{can\_msg} = \text{csp2can}(\ldots);
\]

AM 2 (identical to SWI test)

\[
\text{arc\_label052!LAV\_S\_alarm}
\]

IFM_ARC_HSI

\[
\text{arc2csp( arc\_msg );}
\]

ARINC Driver Layer

\[
\text{arc\_send(\ldots)can\_recv()};
\]

CAN Message Identifier

<table>
<thead>
<tr>
<th>CAN Data Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>28 27 26 25 24 ... 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module ID</th>
<th>System Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>„LAV_S“</td>
<td>„Smk Detection System“</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>„Alarm“</td>
</tr>
</tbody>
</table>

Jan Peleska, Aliki Tsiolakis

18.4.2002
Example: HW/SW Integration using Interface Abstraction (3)

AM 1 (identical to SWI test)

```
ifm_can_hsi
  can_msg = csp2can(...);
  can_send(can_msg);
```

AM 2 (identical to SWI test)

```
ifm_arc_hsi
  arc2csp(arc_msg);
  arc_send(...)
```

ARINC 429 Data Outputs Label 052

| 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | 1 |

PARI TY

<table>
<thead>
<tr>
<th>CABIN</th>
<th>CABIN</th>
<th>CABIN</th>
<th>CABIN</th>
<th>AV_CO</th>
<th>LAV_M</th>
<th>LAV_L</th>
<th>LAV_S</th>
<th>Label 052</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Jan Peleska, Aliki Tsiolakas
18.4.2002
**Test Specifications: Abstract Machine (AM1)**

**AM 1**: Operates on abstract channels - Simulates specific smoke detector behaviour, e.g. detector at LAV_S

- `can_smk_msg.LAV_S!ok` / `setTimer!random`
- `elapsedTimer`
- `can_smk_msg.LAV_S!alarm` / `setTimer!random`
- `can_smk_msg.LAV_S!sensor_fail` / `setTimer!random`
**Test Specifications:**
**Abstract Machine (AM2)**

**AM 2:** Operates on abstract channels - Checks messages generated by SUT in response to specific smoke detector status, e.g. LAV_S

- `can_smk_msg.LAV_S?status` / `setTimer!t`
- `arc_lable052.LAV_S.status`
- `elapsedTimer`
- `error`

Any other status value `s'` leads to an error.
Advantages of Interface Abstraction

- Re-use of test specifications on all test levels — from Software Integration to System Integration Testing
- Unified interface description method on all test levels
- Abstraction from concrete interfaces by means of Interface Modules (IFMs)
- Use of different abstract machines for simulation and testing possible (e.g., AM1 for simulation and AM2 for checking)
- Algorithms for test generation and test evaluation operate on abstract channel events → re-usable on all test levels
A closer look on test scripts

- **Sequential test scripts** used by conventional testing approaches
  - Test cases consist of a list of sequential steps (stimuli for the SUT or checks of SUT responses)
  - Effort for preparation proportional to the test duration
  - Limitations for manually written scripts (<10000 steps)
  - No re-use of any test script on other testing levels
  - Combinatorial effect of interleaved environment stimuli is difficult to specify in sequential script

1. Approach unsuitable for long-term testing
2. Test execution of sequential test scripts not appropriate for parallel nature of SUT and operational environment
A closer look on test scripts (2)

- Test scripts based on timed state machines
  - Each state of the machine specifies
    - a set of possible inputs to the SUT
    - the set of correct SUT outputs
  - Test steps are transitions in the state machine
  - Test scripts based on timed state machines are interpreted in hard real-time by Abstract Machines (AMs)

- Automatic generation of test data during test execution
- Abstract Machines running in parallel for
  - simulating different parallel components
  - checking different aspects of the SUT behaviour
Interface Abstraction using the RT-Tester

Abstract Machine Layer
Communication Control Layer
Interface Module Layer
System Under Test
Example for HW/SW Integration Testing

- **AM** Monitoring/Checking of Messages to FWS and CFDS
- **AM** Monitoring/Checking of AIP Indications in response to Smoke Warnings
- **AM** Simulation of Smoke Detector Behaviour
- **AM** Monitoring/Checking of AIP Indications in response to Smoke Warnings
- **AM** Simulation of Fire Extinguishing Bottles

- **IFM ARINC**
- **IFM SERIAL**
- **IFM CAN**
- **IFM DIGIO**

- **Smoke Detection Function**
- **Cabin Comm. System**
- **ARINC 429**
- **Serial RS232**
- **CAN**
- **Digital I/O**
Conclusion: Advantages of the Approach

- Unified interface description method on all test levels (from SW Integration to System Integration Testing)
- Re-use of test specifications (simulators and checkers) on all test levels
- Automatic generation of test data from timed state machines
- Automatic evaluation of SUT behaviour by means of state machine checkers
- Simple description of complex behaviours by means of networks of abstract machines, each machine describing a single behavioural aspect
Conclusion: Tool support

- The presented test automation concept is supported by the **RT-Tester tool** (developed since 1993 by *Verified Systems International GmbH* in cooperation with *TZI*)
- Simulation and test of time-continuous aspects by integration of MatLab/Simulink
- Open interface for integration of other test tool components (e.g., for GUI testing)
Conclusion: Application Areas

- Testing of Airbus Avionics controllers developed by *KID Systeme*
  - A340-500/600 and A318 Cabin Communication System CIDS
  - A318 smoke detection controller (SDF)
  - A380 controller tests in preparation
- Testing of train control systems and interlocking system components developed by *Siemens*
- Testing of controller for the International Space Station ISS developed by *ASTRIUM*
- Testing of automotive controllers (*Daimler Chrysler*)
Conclusion: Current Research and Development Activities

- Development of hard real-time timed **test engine based on PC clusters** (European research project VICTORIA)
- Development of **test strategies for aircraft controllers** based on test design patterns (VICTORIA)
- **Automatic generation of interface modules** from descriptions of relations between abstract channels and concrete variables or functions
- **Tool qualification** according to RTCA DO 178B for test of specific A318 and A380 controllers (Qualification for Test of A340-500/600 CIDS controller already in progress)