Model-based Testing of Automotive Systems

Eckard Bringmann and Andreas Krämer
ICST’08

Presented by Julia Rubin
on November 21, 2012
Multidisciplinary Business
Supply Chain of Components
Innovation is Driven by Software

Software will determine more than 90% of the functionality of automotive systems
Model-based Development Became Popular

Improves communication with multidisciplinary stakeholders

Improves communication with customers and with suppliers

Reduces costs by early validation via simulation and testing.
Model-based Development Became Popular

Abstract models describing the behavior of the system.
Model-based Development Became Popular

- Detailed software models which consider all software-related aspects
Validation Stages

**MiL**: Model in the loop

Models run in a virtual environment

- *Functional models* – at the system level
- *Implementation models* – at an individual module and a whole system levels
Validation Stages

**MiL**: Model in the loop

**SiL**: Software in the loop

Software runs in a virtual environment, without any hardware
Validation Stages

**MiL**: Model in the loop

**SiL**: Software in the loop

**PiL**: Processor in the loop

Software runs on a target processor or an emulator, but with a proprietary hardware (ECU)
Validation Stages

**MiL:** Model in the loop

**HiL:** Hardware in the loop

**SiL:** Software in the loop

**PiL:** Processor in the loop

---

```c
int main()
{
    bind_sut_inputs_to_vm;
    bind_sut_outputs_to_vm;
    bind_sut_parameters_to_vm;
    while testrun {
        sut_code
        tpt_vm_run_one_cycle
    }
}
```
Validation Stages

**MiL**: Model in the loop

**SiL**: Software in the loop

**PiL**: Processor in the loop

**HiL**: Hardware in the loop

The environment consist of physical components (electrical, mechanical, hydraulic)

Test Rig
Validation Stages

**MiL**: Model in the loop

**HiL**: Hardware in the loop

**SiL**: Software in the loop

**PiL**: Processor in the loop

**Test Rig**

**Car**
Requirements for Testing

1. **Automation** to support multiple intermediate releases due to software hardware co-design
2. **Portability** between validation stages
3. **Systematic test case selection** to ensure coverage and lack of redundancies
4. **Readability** – “lingua franca” for multiple stakeholders
5. “**Closed loop**” behavior – the execution of the test depends on the behavior of the tested system
6. **Real-time behavior**
7. **Continuous signals**
1. Motivation: challenges of in-car software development process
2. Requirements for testing
3. TPT tool for automotive model-based testing
4. Experience report
5. Future work
6. Summary and discussion
TPT for Automotive Model-Based Testing

1. Test language for describing test cases

2. A method to represent a test suite for a system under test (SUT)

3. A set of adapters for development environments at different lifecycle stages
Example: Exterior Headlight Controller

1. A switch with three states: ON, OFF, and AUTO
2. ON: turn on
3. OFF: turn off
4. AUTO: turn on or off depending on the ambient light, acquired by a light sensor with range 0% … 100%
   - when the system starts, turned on if ambient light is below 70% and off otherwise.
   - turn on if the ambient light around the car falls below 60% for at least 2 seconds
   - turn off if the ambient light around the car exceeds 70% for at least 3 seconds
1. Test Language

switch(t) := OFF

switch is OFF

t >= 2.0

after 2s

switch is ON

after 10s

switch is OFF again

t >= 5.0

ambient light

sensor(t) := 80 + 4*noise(t/10)
1. Test Language

More features (hierarchical state machines, junctions, actions on transitions, …) are available.

No dependency on software implementation or ECU ➜ can run on an arbitrary platform, given the appropriate adapter.
1. Test Language

\[ \text{switch}(t) := \text{OFF} \]

\[ \text{switch is OFF} \rightarrow \text{after 2s} \rightarrow \text{switch is ON} \rightarrow \text{after 10s} \rightarrow \text{switch is OFF again} \rightarrow \text{after 5s} \rightarrow \text{lights} \]

\[ t \geq 2.0 \]

\[ t \geq 10.0 \]

\[ t \geq 5.0 \]

\[ \text{sensor}(t) := 80 + 4 \times \text{noise}(t/10) \]
Another Test Case

switch(t) := OFF

switch is OFF

after 2s

t >= 2.0

switch is AUTO

after 1s

t >= 1.0

switch is OFF again

after 5s

t >= 5.0

Headlight signal remains off
Another Test Case

\[
\text{switch}(t) := \text{OFF} \quad \text{switch}(t) := \text{AUTO} \quad \text{switch}(t) := \text{OFF}
\]

\[
\begin{align*}
\text{switch is OFF} & \quad \text{after 2s} & \quad \text{switch is AUTO} & \quad \text{after 1s} & \quad \text{switch is OFF again} & \quad \text{after 5s}
\end{align*}
\]

\[
\begin{align*}
t & \geq 2.0 \\
t & \geq 1.0 \\
t & \geq 5.0
\end{align*}
\]

\[
\text{ambient light}
\]

\[
\text{sensor}(t) := 80 + 4 * \text{noise}(t/10)
\]
2. Representation of a Test Suite: all TPT Tests are Integrated into a Single Test Model

Variation Points
Deriving Test Instances: Classification Trees

Can express *logical constraints* between variants and describe the desired *coverage* in the space of all possible combinations.
Test Process Using TPT

1. Test Case Design
2. Compilation
3. Test Assessment
4. Report Generation
5. Test Execution
Test Process Using TPT

Highly compact byte code, executed by a dedicated TPT-VM virtual machine
Test Process Using TPT

TPT VM executes the tests and communicates with SUT via adapters.

Input and output signals are recorded.
Recorded Test Signals
Test Process Using TPT

Offline using assessment scripts provided for each test.
Test Process Using TPT

Platform independent:
- Test Case Design
- Compilation
- Test Assessment
- Report Generation

Platform dependent:
- Test Execution
Requirements for Testing

1. **Automation** to support multiple intermediate releases due to software hardware co-design

2. **Portability** between validation stages

3. **Systematic test case selection** to ensure coverage and lack of redundancies

4. **Readability** – “lingua franca” for multiple stakeholders

5. “**Closed loop**” behavior – the execution of the test depends on the behavior of the tested system

6. **Real-time behavior**

7. **Continuous signals**
Outline

1. Motivation: challenges of in-car software development process
2. Requirements for testing
3. TPT tool for automotive model-based testing
4. Experience report
5. Future work
6. Summary and discussion
Experience

- Initially developed in Daimler Software Technology Research
  - In cooperation with many production vehicle development projects
- Used by all interior production vehicle projects in Daimler
- Used by car manufactures and suppliers as a basis for acceptance tests
- The idea to concentrate all test scenarios into a single model has been proven to be scalable
Future Work

• Interaction with version and configuration management
• Interaction with test management
• Product families
• Model-based testing on integration levels
  – Relationship to component models
  – Utilization of architectural models
Summary

Multidisciplinary Business

- Software
- Electronics
- System
- Hydraulics
- Mechanics
...

Requirements for Automated Model-based Testing (MBT)
1. Automation, to support multiple intermediate releases due to software hardware co-design
2. Portability between integration levels
3. Systematic test case selection to ensure coverage and lack of redundancies
4. Readability – “lingua franca” for multiple stakeholders
5. Real-time behavior (HiL level)
6. Continues signals

Validation Stages

- MiL: Model in the loop
- HiL: Hardware in the loop
- SiL: Software in the loop
- Test Rig
- PiL: Processor in the loop
- Car

Test Process Using TPT

1. Platform independent
2. Platform dependent
3. Test Case Design
4. Compilation
5. Test Assessment
6. Report Generation
7. Test Execution

Discussion

• Seems to be very useful in practice
• Some important details are omitted, e.g.,
  – How to design and manage assessment scripts?
  – How difficult is it to build adapters?
• Would be interesting to see how scalable the representation of “all test cases in one model” is
• No related work is discussed
  – Most of the provided references are not cited in the text
• What makes the tool specific to automotive?
  – Can it be used in other domains, e.g., A&D
Thank You!
Backup
What is Model-Based Testing?

“In the automotive industry [the term MBT] is normally used to describe all testing activities in the context of model-based development projects”
The V Development Model

Customer’s Requirements -> Specification

System Design

Software Design

Code

MiL

System Test

Software Test

SiL, PiL, HiL

Validation Test

Test Rig

Car
The PikeTec founders Dr. Eckard Bringmann, Andreas Krämer und Dr. Jens Lüdemann share a ten year working experience at DaimlerChrysler. Their research focussed on software technology and tool development for the software development of technical systems.

With their work they supported the development group at DaimlerChrysler AG especially with software development by defining and designing development processes. This support included also the technical and scientific steering of development projects in the automotive divisions passenger compartment, driving dynamics, telematics and powertrain.