Miroslaw Panek began his career at Delphi TCK in 2004 and since then he has been working as Software Systems Test Engineer. Mirek holds M.Sc. degree in Industry Computer Engineering from the University of Science and Technology (AGH) in Krakow.

DELPHI Technical Center Krakow
Electronics & Safety Division
Table of Contents

- Introduction to Automotive Software Systems and Their Development
- Driving Forces for Model Driven Development in Automotive
- Model Based versus Model-less Development
- Test Harness for Automotive Software System Tests
- Model in the Loop, Software in the Loop, Hardware in the Loop
- Tools for Model Based Development and Testing in Automotive
- Matelo Tool Example
- Reactis Tool Example
- Simulink/Stateflow Models Verifiers and SystemTest Tool Example
- Advantages & Challenges of Model Based Testing in Automotive
Introduction

This presentation provides an overview of model based development and testing in automotive based on DELPHI Technical Center experiences.

The main focus of this presentation is model based testing as applied to system testing (functional, black-box testing) other testing levels like unit tests and integration tests are not covered by this presentation.
Vehicle software system is an distributed embedded software system where independent microprocessor systems (Electronic Control Units - ECU’s) communicate together using different communication networks.

- **LIN** – Local Interconnect Network (low-cost, time triggered, one-wire, master-slave, up to 20kbits/s)
- **CAN** – Controller Area Network (CSMA /w arbitration, fault tolerant, dual wire B - 125kbits/s, C - 1Mbits/s)
- **FlexRay** – (time triggered (TDMA), fault tolerant, 2 x 2 wire, up to 10 Mbits/s)
- **MOST** – Media Oriented System Transport (fiber optic ring, synchronous/asynchronous packets up to 25Mbits/s)

Modern middle class vehicle has about 30 different co-operating microprocessors systems.
Areas of Model Based Development in Automotive

Model based development and testing started in automotive in those areas where simulations where intensive:
- first in safety systems
- next in engine control (power-train)
- and in body control
- further in entertainment/multimedia (about to start or just started)

**Power-train** – pretty demanding systems
high real time, multidimensional optimization
around tens of different correlated control functions
based on tens of real time monitored sensor inputs.

**Safety systems** - high reliability, short, deterministic response time,
usually based on several input sensors with couple of controlled actuators/devices.
Airbags, rollover protections, stability control etc.

**Body control** – deterministic protocols, high speed gateway with
hundreds of digital inputs and outputs (with some analog)
Fault tolerant (shortages, overload etc.) and real time OS based.
Lights control, wiper-washer, remote access, alarms, control panels etc.

**Entertainment/multimedia** – intensive human-machine interface applications: navigation,
internet access, wireless telecommunication, TV, radios etc.
Quasi real time or real-time OS based, code intensive applications (millions of LOC).
Driving Forces for Model Driven Development in Automotive

Most of the driving forces for Model Driven Development are in automotive very same like in any other areas of software systems development:

- post release bug fixing costs grow exponentially as shown by Barry Boehm[2]
- requirements often change and are often wrong what is the main reason of late projects[3]
- time to market is pretty crucial for product success

Modeling is present in automotive development since 80s because of simulations. It was very natural path to get from model based simulation into model driven development and testing at 90s.


Reasons for late software as reported by Embedded Market report [3]. Note: Percentages sum to over 100% due to multiple responses.
Model Based versus Model-less Development - Process

- **Requirements**
  - Customer Requirements Elicitation
  - Physical Model
- **Software Developer**
  - Manually Developed Code
  - ECU
- **Code Generator**
  - Automatically Generated Code
  - ECU
- **Tests Generator**
  - Manually Generated Test Specification
  - Automatically Generated Test Scripts
  - Test Bench
  - Tests Developer
  - Manually Developed Test Specification
  - Manually Developed Test Scripts
  - Test Bench

Model Based Development Progress

Model Based Testing Progress
Model Based versus Model-less Development - Costs

Based on [1]

Figure 4: Model-based testing costs
Test Harness for Automotive System Tests

Automatic test case generation is not enough to have useful working test harness for embedded Device Under Tests. Generated tests need to integrate with usually existing test automation environment both software (scripting, TestStand, TTCN-3, etc.) and hardware (physical inputs generation, physical outputs processing etc.).
Inputs applied to the **model** running on the virtual environment (PC)

**Model-in-the-Loop**

Inputs applied to the **final software** running on the virtual environment (PC)

**Software-in-the-Loop**

Inputs applied to the **final software** running on the final hardware

**Hardware-in-the-Loop**

Test Inputs

Test Outputs

Test Outputs

Test Results

Pass/Fail

Results Comparison

≈
UML based modeling is not extensively used in automotive model based development and tests.

**Why?**
- “Because it’s clunky!” (?) ☺ [4]
- UML is far better suitable to software architecture design than to simulation and modeling of physical control systems what is a must in case of embedded software development
- UML can be used to model systems where information is processed but is not very suitable to model systems where both physical signals and information are processed

**Example.**
Microprocessor system needs to filter input signal (1st order low pass filter) to eliminate noise out of it.
Can UML be used effectively to model this behavior of embedded software? Rather not!
MathWorks Simulink can model this with one block from which executable source code can be easily generated.
Tools for Model Based Development and Testing in Automotive – COTS Tools

There are commercial off the shelf tools (COTS) suitable for Model Driven Development and Tests in the embedded software development world.

Examples:

Modeling
- MATLAB, Simulink, Stateflow from MathWorks
- StateMate from IBM Telelogic (formerly I-Logix)

Auto-coding
- Real Time Workshop from MathWorks
- TargetLink from dSpace

System Test Level Test Generators
- SystemTest from MathWorks
- Reactis from Reactive Systems
- MaTeLo from All4Tec

October 2008
MaTeLo® Tool Example - Introduction

MaTeLo from All4Tec[6] is an example of a tool which supports model based testing through the system usage model (functional, black-box testing) without physical system simulation and auto-coding capabilities.

It uses Markov chains to represent so named system usage models.

Markov Chain is a states/event diagram where next step depends on the current state and on a probabilistic process instead of a deterministic one. Markov chain can be represented as a directed graph below[5].

---

Usage model is created by assigning probabilities for the state transitions based on different possible system usage profiles. Usage model represent interaction between system and user it is not implementation model like in StateMate or Stateflow but functional model. Transitions represent user actions performed on the system (stimulus) and system responses to those stimulus.

Once Markov chain model is created a statistical experiment is run by the tool (using simulation Monte Carlo like) to get test cases out of the model as well as to get statistical data from the experiment.

Random path through the system states is generated which is driven by the transitions probabilities and randomly generated inputs which triggers the transitions. Different algorithms are available to generate test suits.
MaTeLo® Tool Example – Overview

- develop usage profiles (normal user, advance user, service etc.)
- identify functional states of the modeled system
- identify transitions between states
- define usage probabilities (frequency of usage)
- associate transitions with system inputs and outputs, define expected outputs (can associate also with TestStand or TTCN-3 test automation tools)
- models are usually hierarchical what is supported by the tool
- perform formal model verification (e.g. is sum of transition probability equal 1 etc.)

Usage Model Editor®
- static model check (visiting probability, test case length etc.)
- test generation strategies (most probable, random, limit, minimum test suites)

Test Suits Generation and Test Campain Analysis

Usage Model Development (Marcov Chain)

Manual Test Cases (XML)

Automatic Test Cases for TestStand

Automatic Test Cases for TTCN-3

Test Campain Analysis Report

Testor®
- usage probability
- reliability
- Mean Time To Failure
- states and transitions coverage etc.

DOORS®

Usage Model Editor® performs formal model verification

Requirement Management

Testor®

Manual Test Cases (XML)

Automatic Test Cases for TestStand

Automatic Test Cases for TTCN-3

Test Campain Analysis Report

Testor®
- usage probability
- reliability
- Mean Time To Failure
- states and transitions coverage etc.

DOORS®

Usage Model Editor® performs formal model verification

Requirement Management

Testor®
- usage probability
- reliability
- Mean Time To Failure
- states and transitions coverage etc.

DOORS®

Usage Model Editor® performs formal model verification

Requirement Management

Testor®
- usage probability
- reliability
- Mean Time To Failure
- states and transitions coverage etc.

DOORS®

Usage Model Editor® performs formal model verification

Requirement Management

Testor®
- usage probability
- reliability
- Mean Time To Failure
- states and transitions coverage etc.

DOORS®

Usage Model Editor® performs formal model verification

Requirement Management

Testor®
- usage probability
- reliability
- Mean Time To Failure
- states and transitions coverage etc.

DOORS®

Usage Model Editor® performs formal model verification

Requirement Management

Testor®
- usage probability
- reliability
- Mean Time To Failure
- states and transitions coverage etc.

DOORS®

Usage Model Editor® performs formal model verification

Requirement Management

Testor®
- usage probability
- reliability
- Mean Time To Failure
- states and transitions coverage etc.

DOORS®

Usage Model Editor® performs formal model verification

Requirement Management

Testor®
- usage probability
- reliability
- Mean Time To Failure
- states and transitions coverage etc.

DOORS®

Usage Model Editor® performs formal model verification

Requirement Management

Testor®
- usage probability
- reliability
- Mean Time To Failure
- states and transitions coverage etc.
Simulink/Stateflow® Based Tools - Introduction

MathWorks[7] provides since years MATLAB and Simulink/Stateflow toolboxes which are used for system modeling and simulation.

Simulink/Stateflow models are an input to both code generators as well as to model checkers and test generators developed either by MathWorks itself but also by third party companies like dSPACE[8] or Reactive Systems[9].

In this approach black-box test cases are generated from same models which are used for simulations i.e. from physical or implementation models.

In addition to black box test cases generation there are also tools available which performs static model analysis as well as can debug and execute models itself (executable models).

In the simulation environment test cases (TC) are defined for a model as an ordered pair of vectors in time with analog and digital values possible on inputs and outputs. Such test cases can be applied to state as well as state-less systems, digital (discrete) and analog (continuous) one as well as to all those mixed variants.

\[
\text{TC} := (I(t_k), O(t_k)), \quad t_0, t_1, t_2, \ldots, t_k
\]

**n-inputs**

\[
I(t_k) = \begin{bmatrix} i_1(t_k) \\ i_2(t_k) \\ \vdots \\ i_n(t_k) \end{bmatrix}
\]

\[
i_j(t_k) = \begin{cases} 1, & \text{for digital inputs} \\ 0, & \text{for analog inputs} \end{cases}
\]

\[
i_j(t_k) \in [i_{j\text{min}}, i_{j\text{max}}], \quad j = 1, 2, \ldots, n
\]

**m-outputs**

\[
O(t_k) = \begin{bmatrix} o_1(t_k) \\ o_2(t_k) \\ \vdots \\ o_m(t_k) \end{bmatrix}
\]

\[
o_j(t_k) = \begin{cases} 1, & \text{for digital outputs} \\ 0, & \text{for analog outputs} \end{cases}
\]

\[
o_j(t_k) \in [o_{j\text{min}}, o_{j\text{max}}], \quad j = 1, 2, \ldots, m
\]
Reactis® Tool Example - Overview

**Coverage Analysis**
- Subsystems, branches, states, conditions, decisions, MC/DC, boundaries, lookup targets, user targets, assertions

**Test Case Generation**
- Generates tests for Simulink and Stateflow models

**Model Translator**
- DOORS®
- MATLAB® / Simulink® / Stateflow®
- Reactis®

**Model Analysis**
- Reactis Validator searches for defects and inconsistencies in models

**Test Execution**
- Reactis Simulator provides single- and multi-step execution, breakpoints. It allows also visual tracking of coverage data.

October 2008
Simulink® Verification and Validation™ Toolbox - Overview

- **Requirement Management**
  - DOORS®

- **Requirement Modeling, Design Capture, Simulation**
  - MATLAB® / Simulink® / Stateflow®

- **Coverage Analysis**
  - Cyclomatic complexity, decision coverage, condition coverage, MC/DC coverage, lookup table coverage, signal range coverage

- **Model Analysis**
  - Checks Simulink and Stateflow models for compliance with modeling standard (e.g., The MathWorks Automotive Advisory Board (MAAB) Style Guidelines)

- **Requirement Traceability**
  - Links requirements stored in DOORS, Microsoft Word, Microsoft Excel, PDF or HTML with Simulink and Stateflow objects, tests, and generated code
Simulink® Design Verifier™ Toolbox - Overview

**Model Analysis**
Detects unreachable design elements in models, proves model properties and generates examples of violations.

**Test Case Generation**
Generates tests for Simulink and Stateflow models.

**Coverage Analysis**
Decision coverage, condition coverage, MC/DC coverage.

**Test Harness**
Harness model allows testing the component in simulation.

**Test Report**
Produces detailed test-generation and property-proving analysis report.

**Requirement Management**
DOORS®

**Requirement Modeling, Design Capture, Simulation**
MATLAB® / Simulink® / Stateflow®
Advantages & Challenges of Model Based Testing in Automotive

**Advantages**
- Faults can be early detected (especially when requirements are expressed through executable models)
- Once model is developed requirements changes can be easily applied to the generated software and tests
- Large amount of test cases can be generated (and executed)
- Product time-to-market can be shortened (if methodology correctly applied)
- Lots of system aspect can be verified and validated before real hardware and software are available
- Test reporting is automatic (with various kinds of metrics generated)

**Challenges**
- Model development (especially useful executable model development) is very time consuming
- Tools impose various constrains on modeling (they need to be well known before used)
- Additional training is required to get team skilled in both model based approach and used tools
- Sometimes that is difficult to compare different metrics generated by different tools
- Generated test cases are not always obvious (and sometimes not realistic what is difficult to identify)
- Effort estimation methods for model based testing are so far not well established
- Generated test cases integration with existing test automation environment is not always obvious and easy
- Keeping bilateral traceability between test cases and requirements is difficult
- No “push a button” approach – detailed tool and model knowledge is needed
- Tools are expensive
References

3. Reasons for late projects, as reported by Venture Development Corporation. Source: Embedded Software Strategic Market Intelligence report, Volume 1, July 2004, VDC
6. All4Tec Main Web Page. [http://www.all4tec.net/](http://www.all4tec.net/)
Thank You!

DELPHI