AVL Embedded Software Model-Based Design Platform
Based on MATLAB and Simulink
MathWorks Automotive Conference 2015
Stuttgart, 24 September 2015
Outline

1. Company use-cases introduction
2. In-house tool platform overview
3. Technical challenges and innovative solutions
Overview AVL Powertrain Controls
Business Fields

Passenger Cars
2-Wheelers
Racing

Construction
Agriculture
Commercial Vehicle

Locomotive
Marine
Power Plants

Software and function development
What is “AVLab”? AVLab (~AVL+MATLAB) describes the tool platform developed at AVL PTE Controls for supporting Model-Based Embedded Software Development in MATLAB and Simulink.
“AVLab” Approach and Philosophy

- Do not make, if you can buy state-of-the-art tool covering our needs
- Build toolchain/interfaces between existing tools (glue tools) to have an *integrated seamless toolchain*
- Don’t be dependent but flexible (*open* platform)
- Cover/support all standard tool landscape/customer scenarios with least effort
- Component-based approach
- Support processes/methods
- Standardization
- Proxy for best practices
- Re-use oriented
AVL Customer Toolchain Use Cases

MATLAB Release?
- Embedded Coder
- Code Generator

Tool chain constraints?

Delivery Scenario?
- White Box incl. Models?
- Only Code sharing
- With ADD?

Customer Guidelines?

Collaboration?
- AVLab Collaboration License
  - With AVLab
  - Pure Simulink
  - Transformations
- No AVLab
- Labels Management
- ADD
- WS mpt

Naming Convention?

AVLab
- TargetLink

TargetLink
- dSPACE
- Embedded Coder
- MATLAB R2008

Customer A
- ETAS
- ASCET-MD

Customer B
- dSPACE
- TargetLink

Customer C
- MathWorks
- Embedded Coder

Customer D
- AVL PTE Controls

Model-based Function and Software development

Modeling
- Physical Model
- Failure point

Code generation
- Implementation Model
- Fixed-point

SW Build
- C Code
- Cross-compiler
- linker / loader

Application Software
- contains the "low-level" control algorithms and calibration data (e.g., inertia-moment models, torque control, knock control, CIS, etc.)
- contains the "higher-level" control algorithms ("drivers")
- contains the operation system

μC evaluation
- μC-Roadmap and performance evaluation

Basic software
- contains "low-level" algorithms
- contains ISO dependent parts of algorithms ("drivers")
- contains the operation system

Basic Software
- contains the operation system
- contains the "higher-level" control algorithms ("drivers")

ECU Hardware
- AUTOSAR RTE
AVLab Timeline

Projects Highlights

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/1/2012</td>
<td>Start AVLab Development (2 people)</td>
</tr>
<tr>
<td>11/19/2012</td>
<td>v1.4 Test Scripts</td>
</tr>
<tr>
<td>6/21/2013</td>
<td>v2.3 Test Evaluation via Reference/Regression Test</td>
</tr>
<tr>
<td>10/1/2012</td>
<td>PMT Worldwide / Established Toolchain</td>
</tr>
<tr>
<td>8/1/2013</td>
<td>AVLab sold to Customer (EC)</td>
</tr>
<tr>
<td>2/8/2013</td>
<td>v2.0 Tester with UI</td>
</tr>
<tr>
<td>3/12/2013</td>
<td>v2.1 ADD One-click Update</td>
</tr>
<tr>
<td>10/31/2013</td>
<td>v2.4 Back to Back (EC). Automatic Assertion via Script. Aggregation</td>
</tr>
<tr>
<td>5/28/2014</td>
<td>Coverage Review</td>
</tr>
<tr>
<td>1/22/2015</td>
<td>Collaboration without ADD/TargetLink</td>
</tr>
<tr>
<td>3/6/2015</td>
<td>v2.7 Batch Framework. Delivery Package</td>
</tr>
<tr>
<td>10/1/2014</td>
<td>PMT Worldwide / Established Toolchain</td>
</tr>
<tr>
<td>10/1/2012</td>
<td>Full EMS for Gasoline Engine (EC)</td>
</tr>
<tr>
<td>10/1/2014</td>
<td>Collaboration Scenarios (TL)</td>
</tr>
</tbody>
</table>

AVLab Team now: ~9 people
AVLab has a strong in-house user basis (188 users).

AVLab supports the MBD development process in several AVL PTE affiliates around the world.
AVLab Modules and Tool Integrations

Data Management

Documentation

ALM Tool

Batch Framework

SyncTool
StiGen
Tester
Code Generation
Consistency Review
Aggregation
Model coverage
Modeling guidelines
Renaming
Delivery
Collaboration

ADD
FunDoc
Integrity
Concerto
TargetLink
Embedded Coder
Simulink VnV
MES MXAM
AVLab Advantages

Before AVLab:
- Too many tools, too complex
- Local project specific solutions/scripts (fill missing feature)
- High Cost and time delay because of tools
- Effort to link tools
- Multiplied maintenance and effort by project
- Developer without guidance

With AVLab:
- Shorter development time
- Increased efficiency and productivity (one-click solutions)
- Better quality
- Best practices proxy/levelling up
- Easy re-use
- **One** standard workflow/**one** way (from start to end/continuity)
- **One** platform (tool linked together/seamless toolchain)
Development Levels and Test Platforms

Pushing Performance Limits using:
- Windows 64bit
- Reference Model
- Accelerated Mode
- Simulation Parallelization

Challenge Heat Map for MATLAB and Simulink

Possible with MATLAB and Simulink

Test Front-loading

MiL floating point
MiL fixed point
SiL
PiL
Virtual Integration Platform
HiL
XiL
Test Bed
Vehicle
List of Challenges

We will present our current answers to some of following questions/challenges:

- **How to ensure Traceability to the System-Under-Test?**
- How to support Data Management for both Embedded Coder and TargetLink?
- How to handle Calibration Data for a component and Test Cases?
- **How to handle simulation data in a lean way to reduce out of memory issues but still ensure test results consistency?**
- How to push the limits of full ASW System simulation on MATLAB and Simulink?
- **How to support component aggregation in MATLAB and Simulink in a semi-automatic way?**
- **How to ensure consistency between all development artefacts?**
- How to ensure test continuity between different test platform (example MiL/SiL -> HiL)? [re-use and Back-to-Back]
- ...
Lean Specification Data management

Test Specification Data are handled in single files.
One directory per Test Case.
Test Cases directory contains
- Specification data (stimuli, calibration file)
- Evaluation data (plot config, assessment script, reference signals)

Advantages:
- Allows direct access to information (example Test Case calibration)
- Straightforward re-use of test cases
- Traceability Test Case Item to test case data (source traces in Integrity) with suspect/impact analysis
Lean Modular Simulation Data management

Simulation Data

- Simulation Data is split from Configuration Data and saved in a separated file
- Simulation Data belonging to a Test Session are packed/ zipped together for test run consistency
- If the System-Under-Test is unchanged (checksum), Data are merged. Else reset.

Advantages:

- Ensure consistency
- Lean memory usage for evaluation
  - Only result data from one Test Case is loaded simultaneously.
  - Only the necessary signals are loaded. (mat-file API)
- Supports Simulation Parallelization
Component Aggregation

Project Interface to build/aggregate components

- Build/update model (using library or model reference)
- Build data files (flat included or referenced)
  - For TargetLink also .dd files are merged
- Build calibration file (flat included or referenced)

Semi-automatic: scheduling has to be finished manually.
Traceability & Consistency Challenge

In automotive SPICE 3.0 special focus on traceability and consistency.

Example: Test Report and Test Results shall reference to the revision of the System-Under-Test.

Source: B. Sechser
http://www.slideshare.net/Polarion_Deutschland/automotive-spice-30-was-ndert-sich
Traceability System-Under-Test (SUT)

Challenge:
Assure all test work products are traceable to their tested objects.
“What was tested?”

Summary of solutions

1. Display revision as Expanded Keywords Properties (Model, ASCII Files) ✗
2. Download revisions from Version Management Repository and run test against downloaded unmodified revisions (redirect path to download location)
3. Tracing revision in working Sandbox/on the fly; marking of weak (=modified) revision traces
4. Solution 3 + automatic repair of weak revision traces
Traceability System-Under-Test (Solution 4)

Solution 4: Run against working files and gather on the fly suspect/weak trace information + checksum information

Repair Weak revision traces

Revision $n+1$
Pseudo Checksum*

Revision $n$
Pseudo Checksum*

weak trace

$1.12$
$1.11$

= ?
yes

Repair weak revision traces

1.12
Revision $n+1$
Pseudo Checksum*

1.11
Revision $n$
Pseudo Checksum*

weak trace

*Pseudo Checksum = Checksum of artefact without revision information
Artefacts Consistency Check

Goal:
- While looking at a collection of artefacts, be able to check if they are consistent together.

Problematic examples/pitfalls:
- Are the test artefacts (e.g. Test Report, Test Results) consistent with the data label definition?
- Was the delivered model tested?
  = Is the test configuration in the delivery consistent with the one used for testing?
- Is the documentation up-to-date = consistent with the delivered model revision?
- Is the Test Report consistent with the deliverables?
- ...

Artefacts Consistency Check - tracing

Action Examples:
- Generate Documentation
- Write Test Report
- Run Test Session
- ADD Update
- Generate Code
- ...

Traces are available as text (in file header) and hidden as file properties.
### Artefact Consistency Check/ Matrix - Report

#### Table 1

<table>
<thead>
<tr>
<th>Artefacts/Trace</th>
<th>MUTRev</th>
<th>fDataRev</th>
<th>fcalRev</th>
<th>ADDProjectID</th>
<th>ADDContainerStatus</th>
<th>ADDContainerVersion</th>
<th>ADDDataType</th>
<th>ADDContainerName</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScrCtri\mdl\ScrCtri_Data.m</td>
<td>1.7</td>
<td>1.2</td>
<td>1.8</td>
<td>0803229.7.1.0</td>
<td>fixed</td>
<td>1.2.0</td>
<td>software</td>
<td>ScrCtri</td>
</tr>
<tr>
<td>ScrCtri\mdl\ScrCtri_Lib.mdl</td>
<td>1.6</td>
<td>1.2</td>
<td>1.8</td>
<td>0803229.7.1.0</td>
<td>fixed</td>
<td>1.2.0</td>
<td>software</td>
<td>ScrCtri</td>
</tr>
<tr>
<td>ScrCtri\mdl_test\ScrCtri_Test.mdl</td>
<td>1.7</td>
<td>1.2</td>
<td>1.8</td>
<td>0803229.7.1.0</td>
<td>fixed</td>
<td>1.2.0</td>
<td>software</td>
<td>ScrCtri</td>
</tr>
<tr>
<td>ScrCtri\mdl_test\ScrCtri_TestReport.docx</td>
<td>1.3</td>
<td>1.1</td>
<td>1.8</td>
<td>0803229.7.1.0</td>
<td>fixed</td>
<td>1.2.0</td>
<td>software</td>
<td>ScrCtri</td>
</tr>
<tr>
<td>ScrCtri\src\ScrCtri_TestReport.docx</td>
<td>1.4</td>
<td>1.1</td>
<td>1.8</td>
<td>0803229.7.1.0</td>
<td>fixed</td>
<td>1.2.0</td>
<td>software</td>
<td>ScrCtri</td>
</tr>
<tr>
<td>References</td>
<td>1.7</td>
<td>1.2</td>
<td>1.8</td>
<td>0803229.7.1.0</td>
<td>fixed</td>
<td>1.2.0</td>
<td>software</td>
<td>ScrCtri</td>
</tr>
</tbody>
</table>

#### Table 2

<table>
<thead>
<tr>
<th>Artefacts/Trace</th>
<th>MUTRev</th>
<th>fDataRev</th>
<th>fcalRev</th>
<th>ADDProjectID</th>
<th>ADDContainerStatus</th>
<th>ADDContainerVersion</th>
<th>ADDDataType</th>
<th>ADDContainerName</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScrMd\mdl\ScrMd_Data.m</td>
<td>1.26</td>
<td>1.19</td>
<td>1.23</td>
<td>0803229.7.2.0</td>
<td>draft</td>
<td>2.5.0</td>
<td>software</td>
<td>ScrMd</td>
</tr>
<tr>
<td>ScrMd\mdl\ScrMd_Lib.mdl</td>
<td>1.26</td>
<td>1.19</td>
<td>1.21</td>
<td>0803229.7.2.0</td>
<td>draft</td>
<td>2.5.0</td>
<td>software</td>
<td>ScrMd</td>
</tr>
<tr>
<td>ScrMd\mdl_test\ScrMd_Test.mdl</td>
<td>1.23</td>
<td>1.18</td>
<td>1.2</td>
<td>0803229.7.2.0</td>
<td>draft</td>
<td>2.5.0</td>
<td>software</td>
<td>ScrMd</td>
</tr>
<tr>
<td>ScrMd\mdl_test\ScrMd_TestReport.docx</td>
<td>1.26</td>
<td>1.19</td>
<td>1.19</td>
<td>0803229.7.2.0</td>
<td>draft</td>
<td>2.3.0</td>
<td>software</td>
<td>ScrMd</td>
</tr>
<tr>
<td>ScrMd\src\ScrMd_TestReport.docx</td>
<td>1.26</td>
<td>1.19</td>
<td>1.19</td>
<td>0803229.7.2.0</td>
<td>draft</td>
<td>2.4.0</td>
<td>software</td>
<td>ScrMd</td>
</tr>
<tr>
<td>References</td>
<td>1.26</td>
<td>1.19</td>
<td>1.26</td>
<td>0803229.7.2.0</td>
<td>draft</td>
<td>2.5.0</td>
<td>software</td>
<td>ScrMd</td>
</tr>
</tbody>
</table>
Outlook / Roadmap

2015: Major Milestones:
- Interface to Integrity Test Management
- Review / Consistency Module

Hot Topics: AUTOSAR, MultiCore Support
Continuity towards HiL, Virtual Integration Platform

2016: Continuous Integration (Review)
Conclusion

AVL Powertrain Controls uses an open standard internal platform to support and improve its model-based software development process. It is purely implemented in MATLAB and Simulink and connects many tools together. It tries to face many challenges for achieving high quality and provide a seamless toolchain with best practice state-of-the-art solutions to its users.
MATLAB/Simulink ADD Interface

Synchronize Tool

- One-click ADD Update (ADD to MATLAB: data synchronization)
- ADD2Simulink: Synchronization ADD<-> Simulink
- Consistency Check
- Seamless Integration

Simulink Model and MATLAB Workspace are in Sync
Data Management – Labels Dictionary

Both TL and SL/EC

_ADD_.dd

_TargetLink_.dd

_Simulink/EmbeddedCoder_.sldd

≥R2014a

_One-click Update_

Data Management – Labels Dictionary

_ADD_.dd

_TargetLink_.dd

_Simulink/EmbeddedCoder_.sldd

≥R2014a

_One-click Update_

_Add_.dd

_Simulink Model

_Workspace

_Code Generation

_Simulink Model

_Code Generation

_C-code_
Calibration Files Handling

- Default Global Calibration File under `mdl_test/<compname>.dcm`
- Tear-down
- Test Case calibration files:
  - `<TCid>_Cal.m` (extracted from `_Data.m`; only `.Value`)
  - `_full.dcm` is rewritten after run for documentation and traceability

=> this allows handling of TC calibration variation as Delta in an M-File
Lean Simulation Data Handling

Unzip

Session File

~Component.mat: Meta Information

~TCxxx.mat: Test Run Data

Select TC

Evaluation

Signal 1
Signal 2
Signal 3
Signal 4
Signal 5

Only result data from one Test Case is loaded simultaneously. With matfile API, only the necessary signals are loaded.
Prerequisite: Standard Component Structure

Model Template with Operating System

Support Closed-loop Test with Plant Model
Component Aggregation (Model)

Scheduling is Semi-automatically created at aggregation
Standard Component File Structure

Configuration Management Plan at Component Level

CM Plan at Project/ ASW Sys Level with 3 levels architecture.
Level 1 = Aggregation Level
System-Under-Test Definition

Model-Under-Test (MUT)

System-Under-Test

Calibration File

Test Case Specification files are handled separately and traced via Source Traces.

Split Data Definition from Values

@AVL.mpt

_Data.m

.ddx

temporary file

ADD

Public System - Under Test

Calibration File

Test Case Specification files are handled separately and traced via Source Traces.
Traceability System-Under-Test (Solution 1)

Display revision information as Expanded Keywords Properties

If file is modified after Check-out/ Get, contained revision information is obsolete. This information cannot be used as consistent trace to the VM repository.
Solution 2: Download SUT files and run test session against them without modifying local version in one batch.

**Drawbacks:**
- Need to download revisions (even if already in Sandbox.)
- Cumbersome path redirect handling.
Traceability System-Under-Test (Solution 3)

Solution 3: Run against working files and gather on the fly suspect/weak trace information

2.2 Test Environment

<table>
<thead>
<tr>
<th>Test evaluation tool</th>
<th>AVlab 2.6.0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation environment</td>
<td>MATLAB/Simulink 8.2 (R2013b)</td>
</tr>
<tr>
<td>MKS Revision</td>
<td>working</td>
</tr>
<tr>
<td>Run Mode</td>
<td>MIL fixed</td>
</tr>
</tbody>
</table>

ADD

<table>
<thead>
<tr>
<th>Container</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>N/A</td>
</tr>
<tr>
<td>Status</td>
<td>N/A</td>
</tr>
<tr>
<td>Data type</td>
<td>N/A</td>
</tr>
</tbody>
</table>

System under Test

<table>
<thead>
<tr>
<th>MUT</th>
<th>LamiSpBas_test.mdl 1.28 (modified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data File</td>
<td>LamiSpBas_Data.m 1.26</td>
</tr>
<tr>
<td>Calibration File</td>
<td>LamiSpBas.dom 1.5</td>
</tr>
</tbody>
</table>

Advantages:
- No need for download and SUT redirect
- Can be checked automatically (at review, checkpoint...)

Drawback: Need to check-in/commit and re-run test for green reporting
Parallel Execution

Parent MATLAB Instance

Run Test Case #4
Run Test Case #2
Run Test Case #3

Parallel Simulation
Example 3 Children MATLAB Instances

Allocated memory in children MATLAB instances is freed up after every run.

Child MATLAB Instance

Run Test Session
Parallel Execution - Process

Parent MATLAB Instance

Save

Load

Start via API

Parent MATLAB Instance

Log file TCXXX

Poll log files

Load incrementally

Log file TCXXX

~TCXXX.mat (result data)

Write status

Save result

Worker MATLAB Instance

Child MATLAB Instance

~Component.mat (meta information)