Model-Based Verification of Automotive Controllers

Rance Cleaveland, PhD

Professor of Computer Science, University of Maryland

and

Executive & Scientific Director, Fraunhofer USA Center for Experimental Software Engineering (CESE)
This Talk

- Model-based validation
  - ... of automotive software product lines
  - ... using instrumentation-based verification

- Talk structure
  - Modeling in automotive software development
  - Instrumentation-based verification
  - Product lines
  - An approach to product-line validation
  - Conclusions
Automotive Software

• Driver of innovation
  
  90% of new feature content based on software [GM]

• Rising cost
  
  20% of vehicle cost [Conti], 50% for hybrids [Toyota]

• Warranty, liability, quality

  High-profile recalls in Germany, Japan, US
A Grand Challenge

• Ensure high quality of automotive software
  – ... preserving time to market
  – ... at reasonable cost

• Key approach: Model-Based Development (MBD)
Traditional Software Development

Requirements / specs / designs / test plans / etc.  

Source code
Model-Based Development

Use models (MATLAB® / Simulink®) as designs / specs
Main motivation: autocode! Also:
- Models support V&V, testing, communication among engineers
- Models can be managed electronically
Simulink®

- Block-diagram modeling language / simulator of The MathWorks, Inc.
- Hierarchical modeling
- Continuous-time and discrete-time simulation
- Used in MBD of control software
Stateflow®

Stateflow chart diagram showing
- Off state with entry: mode = 1
- On state with:
  - Inactive entry: mode = 3
  - Active entry: mode = 4
  - Init entry: mode = 2

Transitions include:
- [onOff == 0]
- [onOff == 1]
- [activate == 1]
- [deactivate == 1]
- [set == 1 && ... deactivate == 0]

Ready (LOCKED)

© 2012 University of Maryland and Fraunhofer USA
Reactis®

A model-based V&V tool from Reactive Systems, Inc.

Tester  Generate tests from models (also C)
Simulator  Run, fine-tune tests
Validator  Validate models / C

Simulink / Stateflow / C  Reactis / Reactis for C  Model / code

© 2012 University of Maryland and Fraunhofer USA
Generating Tests: Guided Simulation

Reactis systematically generates inputs to drive simulation runs to cover model, produce test suites.
Generated Test Data

<table>
<thead>
<tr>
<th>Port</th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
<th>Step 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1: onOff</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2: accelResume</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>3: cancel</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>4: decelSet</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>5: brake</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>6: gas</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>7: inactiveThrottleDelta</td>
<td>0.1</td>
<td>0.0</td>
<td>0.1</td>
<td>-0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>8: drag</td>
<td>-0.0093...</td>
<td>-0.0089...</td>
<td>-0.0094...</td>
<td>-0.0088...</td>
<td>-0.0089...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: active</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2: throttleDelta</td>
<td>-0.1</td>
<td>0.0</td>
<td>-0.1</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td><strong>t</strong></td>
<td>0.0</td>
<td>1.0</td>
<td>2.0</td>
<td>3.0</td>
<td>4.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialSpeed</td>
<td>15.79179838897</td>
</tr>
</tbody>
</table>
Ongoing Research

Design-time modeling, requirements verification
Instrumentation-Based Verification

- Model-validation technique supported by Reactis
- Combines assertions in models, testing
Instrumentation-Based Verification: Requirements

- Automatic verification requires formalized requirements
- IBV: formalize requirements as \textit{monitor models}
- Example
  “If speed is $< 30$, cruise control must remain inactive”
Instrumentation-Based Verification: Checking Requirements

- Instrument design model with monitors
- Use coverage testing to check for monitor violations
- Reactis:
  - Separates instrumentation, design
  - Automates test generation
• Three-month case study with Tier-1 automotive supplier on production system

• Artifacts
  – 300-page requirements document
  – Some source code

• Results (intern)
  – 62 requirements for 10 design features formalized as monitor models
  – Requirements checked on feature models
  – 11 inconsistencies in requirements identified
  – Key technical insight: architecture for monitor models
“[This] is the complete description of the control of the CAN output signals can1 and can2 produced by Function A. Function A can be activated only with \( in = 1 \). The activation takes place when either the CAN bus messages a or b is present....”
Final Monitor Model Architecture

Need for *conditional requirements*

– Behavior only specified for certain situations

– “If timeout occurs do something”
Software Product Lines

• (From SEI): product line = “a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way”

• Key terms
  – Common assets
  – Variation points
  – Variants
SPL in Automotive

• Toyota: 1,800 variants for engine control software
  – Diesel vs. gas vs. hybrid
  – Different emissions regulations
  – Performance profiles for different markets
  – # of cylinders
  – Cruise control?
  – Etc.

• Product lines offer a framework for streamlining development, maintenance

• What about V&V?
Variants in Monitor Modeling

• Fine-grained product-line info often captured at model level

• How can functionality of product-line models be verified?
  – Want to re-use verification effort
  – Some requirements are *universal* (apply to every variant)
  – Others are *variant-specific*
Example: Cruise Control

- Product line could include following variants
  - Maximum-speed restriction or not
  - Adaptive or not
  - Manual or automatic transmission

- Sample universal requirement
  
  If the brake pedal is pressed, the cruise control shall become inactive.

- Sample variant-specific requirement
  
  If the transmission is manual, then the cruise control shall become inactive if the desired speed is inconsistent with the current gear.
How To Do V&V for Product-Line Models?

• Use IBV!

• Result of industrial study
  – Framework for modeling product lines in Simulink
  – Strategy, architecture for variant-specific monitor-models
  – Use of IBV to debug models, find requirements issues
Product-Line Modeling

• Model file defines control functionality
• Configuration file defines parameters
• Some parameters used to define which variant is intended

Model file
“if num_cyl = 4 ...”

Config file
“num_cyl = 6;”
Pilot Study: Cruise Control

- Simulink model is in Reactis distribution
- Partner adapted it as sample product-line model
- Variants
  - Max-speed limitation
  - Adaptive
  - Manual vs. automatic transmission
  - Output interface
Finalizing Product-Line Model in Simulink / Stateflow

• Program variant selection
  – Introduce parameters into model
  – Define MATLAB variables for use as parameters

• Product line contained in two files
  – cruise_variants.mdl (model)
  – cruise_constants.m (MATLAB variables)
Parameterized constant
MATLAB variable

VARIANT_TRANSMISSION
VARIANT_TRANSMISSION
VARIANT_TRANSMISSION
VARIANT_TRANSMISSION

MANUAL
AUTOMATIC
NUM GEARS
GEAR_TABLE
MINIMUM
MAXIMUM

MATLAB variable
Variant-Specific Monitor Models

• Idea
  – Configuration files define variant-selection parameters
  – Why not refer to same parameters in monitor models to introduce variant-specificity?

• Pilot study
  – Defined six example variant-specific requirements
  – Translated each into monitor model
Example

[MS1] If the maximum-speed limitation is enabled, the cruise control shall not permit the desired (set) speed to exceed a designated maximum value.
Monitor Model Logistics

- Monitor models stored in single Simulink library file
- Monitor models refer to parameters

```
cruise_variants.mdl
Product-line model
```

```
cruise_variants_monitors.mdl
Monitor models
```

```
cruise_constants.m
Parameter file
Include variant-selectors
```

instrumented by

reads

reads
Verification

- Product-line model instrumented with monitor models
- Coverage testing used to check for violations
- Reactis® used for both tasks
Verification Results

• Bugs found in product-line model (fixed)
• Bugs found in monitor model (fixed)
• Variant-interaction problem discovered
  – One variant specified maximum speed
  – Other variant specified speed-control by adaptive mechanism
This Talk

Model-based verification of software product lines

– Model product lines in Simulink / Stateflow
– Variant specificity in monitor models
– Instrumentation-based verification
– Variant interactions!
Larger Issues

• Single models vs. parameterized models
  – Typical problem: find parameter settings that ensure satisfaction of requirements
  – Here: parameterize requirements, check consistency of parameterized models vis a vis parameterized requirements

• Parameter interactions

• Requirements are not the always what’s required
Thank You!

Rance Cleaveland
University of Maryland / Fraunhofer USA CESE
rance@cs.umd.edu
+1 301-405-8572