Model-Driven Design and Validation of Embedded Software

Giuseppe Di Guglielmo†
Masahiro Fujita†
Cristina Marconcini‡
Andreas Foltinek•

Luigi Di Guglielmo*
Franco Fummi*
Graziano Pravadelli*

† VDEC
University of Tokyo
Japan
‡ STM Products srl
Italy
• IMACS GmbH
Germany
* Dep. Computer Science
University of Verona
Italy
Outline

• Motivations
• State of Art
• Design and Validation framework
  – radCASE
  – radCHECK
• Comparisons
• Conclusions
Motivations

Specification

(by means of natural language)

ESW design

Manual code definition

Validation

Direct interaction of architects, designers and verification engineers

Model-driven Design

Model-driven Validation

6/15/2011

AST 2011
State of Art (I)

- **Model-driven Design** \((MDD)\)
  - Simplifies the modeling by raising the abstraction level of the description
    - Use of generic models for architecture and behaviors
    - Generic in the choice of coding languages
    - Generic in the choice of execution platforms
  - **Advantages:**
    - Most suitable solution to develop complex systems
  - **Drawbacks:**
    - Need of a validation framework
State of Art (II)

- **Model-driven Validation** ($MDV$)
  - Aims at guaranteeing that the described system models correctly the requirements
  - It may combine
    - Automatic generation of the testcases
    - Assertion-based verification
State of Art (III)

• Automatic generation of the testcases
  – Concrete execution
    • Pseudo-random techniques, genetic and probabilistic algorithms
  – Symbolic execution
    • As a “better” concrete execution
    • Exhaustive reasoning along paths
  – Advantages:
    • Avoid the time consuming process of manual test writing
    • Eases the adoption of dynamic assertion-based approaches
  – Drawbacks:
    • Corner cases may escape if the generation algorithm is not effective
State of Art (IV)

- **Assertion-based Verification (ABV)**
  - Speeds up the detection and the debug of the system errors by using properties
    - Properties are the formalization of the system requirements
    - A property falsification notifies an error into the model
  - **Advantages:**
    - Enables the detection of corner cases errors
  - **Drawbacks:**
    - Complexity of formal languages requires qualified engineers for the property definition process
Goals

• Comprehensive framework for the design and validation of ESW that exceeds the limitations of current Model-driven approaches providing the user with
  – **radCASE**
    • an effective UML-modeling and code-generation environment
  – **radCHECK**
    • an effective validation environment that includes guided property definition and automatic testcase generation
Overview

ESW Informal Specifications and Requirements

radCHECK: Assertion-based Validation
- Property Editor
- Testcase Generator
- Checkers
- Testcases

radCASE: Model-driven Design and Simulation
- Model Editor
- Simulation

Failed requirements

Iterative refinement

Manual activity
- Automatic activity
radCASE

• radCASE: model-driven design and simulation
  – The ESW model is described by means of UML diagrams and C-code
  – The code implementation is automatically generated
    • Support for different target platforms
  – A graphical simulator allows the manually validation of the generated code
radCHECK

- radCHECK: assertion-based validation
  - Property definition
  - Checker generation
  - Testcases generation

- Automate the validation phase
radCHECK: Property definition (I)

• A property editor guides the formalization of the informal requirements into properties
  – Property templates
  – PSL grammar rules

• Property Specification Language (PSL)
  – PSL semantics that allows to formalize complex temporal behaviors in a precise and concise manner
radCHECK: Property definition (II)

• Example of use of a parametric template
radCHECK: Checker Generator (I)

• A checker implements the behavior described by a property
  – It monitors the evolution of a set of variables verifying that their values do not violate the specified property

• A checker consists of C-code executed in parallel with ESW implementation
radCHECK: Checker Generator (II)

• The checker generation
  1. Parsing the property for generating a tree structure that represents the dependences between the property sub-formulas
  2. Mapping each sub-formula into an internal Boolean variable
  3. Mapping each temporal and logical operator into a function that implements the operator meaning
  4. Combining these functions according to the tree structure to model the whole property behavior
radCHECK: Testcases Generation

Initial state

ESW state space

- Corner-case state
- Concrete execution
- Symbolic execution
- Guided Symbolic execution

Missing state due to wide-width search limitation
Tools Comparison

- **radCASE**
  - Comparison with some of the most-used UML model-driven tools in the field of ESW design

- **radCHECK**
  - Property editor
    - Novel idea
  - Checker generator
    - IBM FoCs
  - Automatic testcases generator for ESW
## Model-driven Design Tools Comparison

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature</th>
<th>radCASE</th>
<th>ARTiSAN</th>
<th>Enterprise Architect</th>
<th>Rapsody</th>
<th>VisualSTATE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Requirements</strong></td>
<td>Use Case</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td><strong>ESW structure</strong></td>
<td>Class Diag.</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Object Diag.</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Component Diag.</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td><strong>ESW behavior</strong></td>
<td>State machines</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Sequence Diag.</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Activity Diag.</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>FBDS</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>C/C++</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td><strong>Validation</strong></td>
<td>Simulator</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td><strong>GUI</strong></td>
<td>HMI Editor</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## Checker Generators Comparison

<table>
<thead>
<tr>
<th>PSL operator</th>
<th>CG</th>
<th>IBM FoCs</th>
<th>PSL operator</th>
<th>CG</th>
<th>IBM FoCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>next, next_a, next_e</td>
<td>Y</td>
<td>Y</td>
<td>until!</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>next!, next_a!, next_e!</td>
<td>Y</td>
<td>-</td>
<td>eventually!</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>next_event</td>
<td>Y</td>
<td>Y</td>
<td>always</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>next_event!</td>
<td>Y</td>
<td>-</td>
<td>logical operators</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>next_event_a</td>
<td>Y</td>
<td>-</td>
<td>SERE suffix implication</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>next_event_a!</td>
<td>Y</td>
<td>-</td>
<td>SERE consecutive rep.</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>next_event_e</td>
<td>Y</td>
<td>-</td>
<td>SERE non-consecutive rep.</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>next_event_e!</td>
<td>Y</td>
<td>-</td>
<td>SERE goto repetition</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>before</td>
<td>Y</td>
<td>Y</td>
<td>SERE or</td>
<td>Y*</td>
<td>Y*</td>
</tr>
<tr>
<td>before!</td>
<td>Y</td>
<td>-</td>
<td>SERE length match. And</td>
<td>Y*</td>
<td>Y*</td>
</tr>
<tr>
<td>until</td>
<td>Y</td>
<td>Y</td>
<td>SERE non-len. match. And</td>
<td>Y*</td>
<td>Y*</td>
</tr>
</tbody>
</table>
Conclusions

- Combines the advantages of both MDD and MDV
- Exceeds the limitation of MDD and MDV

Comprehensive
On-line References

• radCASE
  http://www.stm-case.com

• radCHECK
  http://www.verificationsuite.com