Merging OMG Standards in a General Modeling, Transformation, and Simulation Framework

Eighth EAI International Conference on Simulation Tools and Techniques (SIMUtools 2015)
August 24-26, 2015

Vitali Schneider, Anna Yupatova, Winfried Dulz, Reinhard German
Main ideas of Test-driven Agile Simulation (TAS)

- Quality improvements during system development process based on models
- Model-driven techniques for system development, simulation and testing
- Independent, parallel design of system and test models
- Early validation of involved models
  - early testing by means of simulation
  - testing to validate simulated system models
  - simulation to validate test models
- Based on common OMG standards (UML, SysML, MARTE, UTP)
Test-driven agile simulation (overview)
Modeling using OMG standards: UML, MARTE, SysML, and UTP

System Structure
SysML

Modeling using OMG standards:
UML, MARTE, SysML, and UTP

System Behavior
MARTE + ALF

Test Suite
UTP
Transformation using OMG standards: MOFM2T, QVT

- **System Specification**: SysML + MARTE + ALF
- **Traceability of Requirements**: SysML
- **Test Suite**: UTP

**MOFM2T**

**Traceability Model**

**Artifacts** (Simulation Code)

**Artifacts** (Executable Test Suite)
SimTAny tool environment

Modeling

Transformation

Simulation

Analysis

Traceability
Concept of test-driven agile simulation

- **Requirements Engineering**
- **Modeling**
- **Transformation**
- **Simulation**
- **Implementation and Test**

**System Model**

**Simulation Model**

- **static validation**

- **dynamic validation**

**Requirements**

**Usage/Test Model**

**Abstract Test Suite**

**Executable Test Suite**

**Validation**

**System**

**test**
**DSL vs. UML**

- **Domain Specific Language**
  - specific for an application domain
  - simple but completely new language
  - requires special tools
  - high development costs

- **Unified Modeling Language**
  - not domain-specific
  - common and powerful modeling language
  - reuse of existing tools
  - extendable via profiles

→ **UML as modeling language for test-driven agile simulation**
Challenges

- Combination of specialized UML profiles:
  - SysML (Systems Modeling Language)
  - MARTE (UML Profile for Modeling and Analysis of Real-Time and Embedded Systems)
  - UTP (UML Testing Profile)

- Variety of elements

- Different modeling methods

- Syntactic and semantic interferences
  - e.g. similar stereotypes in different profiles
  - e.g. different modeling concepts for similar aspects

→ Modeling methodology is required
Modeling methodology

- **Modeling guidelines**
  - Strategy for combined usage of profiles
  - Reduction of allowed alternatives
  - Unambiguous syntax and semantic

- **Modeling process**
  - Views:
    - requirements modeling
    - system modeling
      - functional model
      - architecture model
      - HW/SW allocation
    - test modeling
  - Abstraction levels:
    - high-level specification
    - refined specification
## Combination of standard UML profiles

<table>
<thead>
<tr>
<th>Modeling aspects</th>
<th>SysML concepts</th>
<th>MARTE concepts</th>
<th>UTP concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>requirements</td>
<td>requirement, trace relations</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>non-functional requirements</td>
<td>requirement</td>
<td>NFP¹ constraints + VSL²</td>
<td>-</td>
</tr>
<tr>
<td>system architecture and allocation</td>
<td>block, part, allocation</td>
<td>structural component, part,</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HW/SW patterns and allocation</td>
<td></td>
</tr>
<tr>
<td>interaction</td>
<td>flow, port, connector</td>
<td>flow, port, flow specification</td>
<td></td>
</tr>
<tr>
<td>value types</td>
<td>value types, unit + dimension</td>
<td>NFP* type, unit conversion</td>
<td>-</td>
</tr>
<tr>
<td>system behavior</td>
<td>common UML concepts + rate, probability,</td>
<td>common UML concepts + time,</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>continuous</td>
<td>performance, probability</td>
<td></td>
</tr>
<tr>
<td>tests</td>
<td>test case</td>
<td>-</td>
<td>test case, test context, test</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>component, test data pool</td>
</tr>
</tbody>
</table>

¹ NFP – Non-Functional Properties  
² VSL – Value Specification Language
Modeling methodology within the development process

- Requirements
  - High-Level system design
    - transform
    - validate
    - refine
  - High-Level simulation
    - transform
    - validate
  - High-Level test design
    - transform
    - validate
    - derive

- System Test
  - integrate

- Acceptance Test
  - integrate

- System/Task Level
  - Block/Instruction Level
    - integrate

- Refined system design
  - transform
  - validate
  - derive

- Refined co-simulation
  - transform
  - validate
  - derive

- Refined test design
  - transform
  - validate
  - derive

- Unit Test
  - integrate

- HW/SW implementation
  - derive
  - integrate
  - integrate
High-level modeling using OMG standards: SysML, MARTE and UTP
Refined specification using OMG standards: SysML, MARTE and UTP

ImageProcessingPipeline

+ EdgeDetection  + ObjectDetection

HW_Architecture_Server

<<hwProcessor>> (architecture="(16,bit)", nbCores="1", nbFPUs="1", mips="(source=calc)") + GPU

<<hwMemory>> (memorySize="(64,MB)", adressSize="(16,bit)", throughput="(source=calc)") + memory

<<hwBus>> (isSynchronous="true", adressWidth="(16,bit)") + bus

<<hwSensor>> + display

<<allocate>> (nature=spatialDistribution)
Traceability

- **Typical questions:**
  - Have all requirements been considered during system/test design?
  - What is the purpose of this part of model – are there any requirements for that?
  - Which elements are affected by a modification?
  - Which tests have to be performed again after a modification?
  - Which requirements are violated if a test case fails?

- **Requires bidirectional traceability**
  - requirements ↔ system model elements/tests
  - system model elements/tests ↔ generated simulation code (artefacts)
  - artifacts ↔ results

- **Modeling of requirements and relations with SysML**
- **Traceability model to store trace information**
- **Traceability view to explore trace information**
Traceability

Simulation code

M2M

QVT

Traceability model

M2T

Simulation code

Acceleo

Relations between model elements

<<block>> Client

<<block>> Camera

<<TestContext>>

TestSuiteClient

<<testCase>>

Check image format

<<verify>>
Traceability in the VeriTAS framework
Conclusions

- **Modeling methodology for TAS**
  - based on standards
  - combination of specialized profiles: SysML, MARTE and UTP
  - iterative modeling process

- **Traceability**
  - requirements ↔ system model elements/tests ↔ simulation code ↔ results
  - generation of trace information from models
  - visualization of trace information
  - connection to modeling tool, test component and resources
Outlook

- Evaluation and advancement of modeling methodology
  - based on the ‘use case’ of image processing systems
    - specification and validation of system architecture
    - analysis and comparison of alternative designs
  - integration of a standardized action language
    - Action Language for Foundational UML (ALF) (OMG standard since 2013)
    - replacement of in-house language “Casual”

- Further development of traceability component
  - completeness of test coverage
  - traceability of found errors back to the requirements
  - support of regression tests
Thank you for your attention!

Questions?
Modeling of test-suites

- Conforming to **UML Testing Profile** (UTP):
  - test context
  - test configuration
  - test component
  - SUT (system under test)
  - test cases describe the interaction with the SUT
Design of usage models

- Model describes the possible usage of the system or a single unit under test
- Either as activity diagrams or state charts on higher abstraction level
- Sequence diagrams for concrete interactions with the SUT
  - Test steps
  - Verification points
Visualization and editing of usage models*

**Visualizer**
- representation of usage models with focus on different test aspects
- extensible for future perspectives
- highlighting of elements (e.g. test case path)

**MCUP Editor**
- creation of usage profiles
  - by definition of probabilities on branches
  - by mixing of existent profiles
  - by placing focus levels
- consistency check and correction of errors

*Cooperation with sepp.med gmbh*
Static validation

- Validation perspective to check the models and display errors
- Validation constraints for checking consistency of corresponding models
- Extensible by additional validation constraints
- List of problems displayed in the problems view
- Erroneous elements highlighted in the model
- **Abstract test cases**
  - are generated by deriving paths from model
  - using various generation strategies
  - can be exported to each compatible test context

- **Model-level test cases**
  - UML interactions
  - referenced interaction fragments of test steps and verification points are concatenated

- **Simulation-level test cases**
  - executable C++ code
  - can be executed with simulation by OMNeT++
Dynamic validation
Simulation and execution of test cases

- Test context (relevant part of system model and test cases) is then executed on simulation level (OMNeT++)
- Simulation results are calculated and plotted by the R tool
- After execution the verdict of each test case is displayed in the test view
Merging OMG Standards in a General Modeling, Transformation, and Simulation Framework

Information flow in a distributed development process