Automated Requirements Verification Model Composition in ModelicaML

Using ModelicaML Value Bindings

Wladimir Schamai, EADS Innovation Works
February 8, 2012
Table of Contents

• Introduction
• Example
• Problem Definition
• Value Bindings Concept
• Verification Models Composition
• Conclusion
Introduction

- vVDR (Virtual Verification of Designs against Requirements) is a method that enables a model-based design verification against requirements.

- ModelicaML integrates a subset of the UML and the Modelica language.

- vVDR can be applied using ModelicaML.

- This presentation discusses the necessary enhancements of the vVDR method, and its implementation in ModelicaML, in order to facilitate automation and, hence, to reduce modeling errors, time and effort.
Introduction: vVDR Method Steps

1. **Formalize Requirements**: This step explicates how select and formalize requirements for the design verification purpose.

2. **Select or Create Design Model to be Verified against Requirements**: This step clarifies what properties a system design model needs to have for being suitable for this method.

3. **Select or Create Verification Scenarios**: This step describes what the properties of a verification scenario are.

4. **Create Verification Models**: This step explains what a verification model consists of.

5. **Execute Verification Models**: This step puts requirements on the simulation output.

6. **Analyze Results**: This step provides guidance for analyzing possible causes of inconsistencies found in the simulation results.
Example: System Design

Simplified Aircraft Potable Water System
Example: Textual Requirement

Filling time: "The time to fill an empty tank shall be 300 sec. max."
Example: Scenario

Scenario: **Filling and Draining**

- **Integer mode(..)**
- **Real pumpPowerFactor**
- **Real preselectedLevel**
- **input Real tankHeight = 1**
- **parameter Real ambPressure(..) = 101325**
- **Integer overflowValveStuckAt = 0**
- **Integer fillDrainValveStuckAt = -1**
- **Integer supplyValveStuckAtPosition = 100**
- **Scenario: Filling and Draining**
Example: Mapping Scenarios to Requirements

- The information about which scenarios are appropriate to verify which requirements is captured by setting dependencies between them.
Problem Definition

- Multiple models become dependent on each other in a specific combination
- What is necessary to determine if and what data a model needs from other models?
- What is necessary to determine which other models can provide the data required by a model?
- It is possible to bind all models and their components automatically?
Solution Proposal

• Each model that requires data from other models should express this need by creating a new mediator or by subscribing to an already existing one.

• Each mediator must have defined providers so that the correct binding code for the clients can be derived.
Example of Value Bindings

Value Bindings

'ModelicaMLModel' model value mediators(1)

- Design: Parameters(4)
- FAILURE Case: Fault Injection(3)
- Requirements: Inputs(8)
  - all valves are fully closed - mediator, Boolean, clients (1), providers (1)
  - fill/drain valve is fully open - mediator, Boolean, clients (1), providers (1)
  - overflow valve is fully open - mediator, Boolean, clients (1), providers (1)
  - system is not used (i.e. not being filled or drain drained) - mediator, Boolean, clients (1), providers (1)
  - tank is being filled - mediator, Boolean, clients (2), providers (1)
  - tank is being trained - mediator, Real, clients (1), providers (1)

- tank is empty - mediator, Boolean, clients (2), providers (1)

- systemIsEmpty - req. client, Boolean, ModelicaMLModel::Requirements::Tank drain
- tankIsEmpty - req. client, Boolean, ModelicaMLModel::Requirements::Tank filling time

- level - provider, Real, ModelicaMLModel::Design::Simple Potable Water System Pack
- tank is filled - mediator, Boolean, clients (1), providers (1)

Test case: Points Of Control(5)
Example of Bindings Traceability

Given a design alternative model:

- **Requirements view** shows what requirements are addressed and can be tested using which scenarios

- **Scenarios view** shows appropriate scenarios and all requirements can be tested with it
Binding Operations: Client Operation

- Used when a single instance must be referenced explicitly, i.e., when the actual client is a sub-component.
- When a value manipulation or type conversion is required
- For overwriting of binding definitions. Any upper binding definition in the components tree overwrites lower level bindings.
Binding Operations: Mediator Operation

- Built-in functions for taking the minimum or the maximum value, sum values, build a product, or to use the logic operators such as AND, OR, or XOR.
- These functions are only used if multiple providers are anticipated for this mediator.
- In contrast, the function getSingleProvider() is used to ensure that there must be only one provider for this mediator.
Binding Operations: Provider Operation

- Used when the actual provider is a sub-component, for example:
  - Requirement: *When the system is not used (e.g. is not being filled, drained or is not supplying water to clients) all valves shall be fully closed.*

- Type conversion, for example, if client requires a different type compared to the provider type:
  - The binding operation `providerPath < 0.001` converts a Real to a Boolean value
Binding Code Derivation

• Based on the model references and the binding operation for clients, mediators and providers, the binding code can be derived automatically when the clients and the providers are in the same components tree.

Binding code for the client `tankIsEmpty` of the requirement R001 instance is

`tankIsEmpty = sm_spws_environment.spws.tank.level < 0.001`. 
Verification Model Composition

- Using Value Bindings we can now determine valid combinations of scenarios and requirements for a selected system design alternative model.
Verification Model Simulation and Results

- The generated Verification Report contains:
  - Test configuration
  - Pass/fail of each test
  - Bindings between models
Conclusion

• This approach enables the definition and reuse of bindings for the on-demand binding code generation and an automated model composition.

• ModelicaML bindings are defined at class attribute level and use the ModelicaML unique identifiers for storing the relation between clients, mediators and providers. The binding definitions are not affected by the change of client and provider names or model structure except when binding operations are used that include references to subcomponents with concrete names.

• Binding definitions do not modify client or provider models (crucial when libraries are used).

• The binding code can be generated on-demand when instantiated models need to be bound, for example in a verification model.

• The generated verification models thereby become artifacts that are created on-demand and do not need to be maintained.
Thank you for your attention!

Wladimir Schamai
EADS Innovation Works (Germany)

wladimir.schamai@eads.net