Automating Model Composition for Design Verification

Wladimir Schamai (Airbus Group Innovations)
Lena Buffoni (Linköping University)
Peter Fritzson (Linköping University)
Daniel Bouskela (EDF)

MODPROD 2015
Problem Description

• Complex physical systems exhibit large numbers of
  – Dynamic continuous states: Physical states (temperature, pressure, mass flow…)
  – Dynamic discrete states: Operating modes (normal, dysfunctional, I&C…)
• They are subject to large numbers of requirements
  – And need many test scenarios for proper verification

• Challenge: Possible combinatorial explosion of situations to be explored
• Question: How to efficiently verify system designs against requirements?
Solution Proposal

- Automate as much as possible the testing of complex systems
  - Formalize natural-language requirements into monitor models
  - Create executable model of the system behavior
  - Use the requirements models as observer of the behavioral model to automatically detect possible violations of the requirements during simulations
  - Use a generator of test scenarios* (to stimulate the design model) that comply with test coverage criteria

\[ \Delta P(t) = k / \rho \cdot Q^2(t) \]

*not addressed in this presentation
Example of Informal Requirements

- ‘When they are in operation, pumps shall not cavitate’.

- ‘The water temperature shall not vary by more than 10°C per hour’.

- ‘When the system is in operation, there should be no less than two pumps in operation for more than 2 seconds. Violation shall not occur more than 3 times per year with a confidence rate of [a given percentage]’.

- …
Formalizing Requirements

- Each individual requirement is formalized into a \textit{violation monitor} model that indicates at any simulated time:
  - untested (default), satisfied, violated

- Modeling language: a new Modelica library with constructs adapted for requirements formalization

- Effort for formalization:
  - Typically a fraction compared to system designs modeling effort

- Added value:
  - Executable monitor models which can be used in simulations, or monitoring of a system in operation
  - Detection of impreciseness or errors in the requirement statement
Model Dependencies

System design model alternatives/versions

Requirements violation monitors

Scenarios (system model stimuli)

Additional computation models
Challenge: Integration of Models

- Models are created independently
  - By different persons
  - In different points in time in system development

- Models cannot/should not be modified, e.g. because they are
  - Library components
  - Black-boxes
  - Ownership, missing knowledge about model details, etc.

- Which models will need to be integrated is not known in advance
  - It is often not possible/practical to agree on fixed interfaces in advance
**Requirement, Design and Scenario**

**Requirement 1: Violation Monitor**
“At least 2 pumps shall be in operation at any time.

```plaintext
model Req
  input Integer numberOfOpPumps = 0 "number of operating pumps";
  constant Integer minNumberOfOpPumps = 2 "min. number of operating pumps";
  output Integer status(start=0, fixed=true) "indication of requirement violation, 0 = not evaluated, 1 = not violated, 2 = violated";

algorithm
  if numberOfOpPumps < minNumberOfOpPumps then
    status := 2 "2 means violated";
  else
    status := 1 "1 means NOT violated";
  end if;
end Req;
```

**Verification Scenario 1**

```plaintext
model Scenario
  output Boolean pump1active "turn on pump1 = true, turn off pump1 = false";
  output Boolean pump2active "turn on pump2 = true, turn off pump2 = false";
  output Boolean pump3active "turn on pump3 = true, turn off pump3 = false";

algorithm
  if time > 0 and time <= 10 then
    pump3active := false;
  elseif time > 0 and time <= 20 then
    pump2active := false;
  elseif time > 0 and time <= 30 then
    pump2active := true;
  else
    pump1active := true;
    pump2active := true;
    pump3active := true;
  end if;
end Scenario;
```

**System Design Alternative 1**
Verification Model Simulation

Question: What is necessary to enable generating the binding expressions automatically?
Composition of Verification Models

Use Case:
- User selects the **system design model** to be verified against requirements
- Tool identifies **scenarios** that can stimulate the selected system model
- Tool identifies **requirements** that are implemented in the selected system model
- Tool composes **verification models**, generates code, simulate, generates report.
Questions

• For a given model: How to bind components?
  – What is necessary in order to determine which components need to be bound?
  – How to generate correct binding expression (=glue code) automatically?

• For an automated/guided verification model composition approach: How to find valid combinations of models?
  – E.g., design verification: For a selected system model, how to find requirements that are implemented and can be tested, and scenarios that can be used to stimulate the system?
Binding: Basic Idea

- The *binding* concept enables capturing of potential interaction between models
  - Some models require data: *Clients*
  - Some models can provide the required data: *Providers*
  - However, clients and providers *do not know each other a priori*

- *Mediators* captures many-to-many relation between clients and providers
  - Multiple clients may require the same data
  - Data from several providers may be needed in order to compute data required by one client
**Binding: Why Mediators?**

- Assume c1 and c2 both require data from p1 and p2 for

![Diagram showing data flow between clients and providers without mediators]

- Group data required by several clients
  - Enables reuse of existing mediators, allows using array reduction functions
  - More concise and consistent

![Diagram showing data flow between clients, mediator, and providers]
Binging Specification Example

- **R1** When the BPS is under maintenance, it must not be activated.

- **R2** In the absence of any BPS component failure or in the presence of a single BPS sensor failure, when the BPS is not under maintenance, and in case of loss of MPS:
  - ...

- **R3** In the absence of any BPS component failure or in the presence of a single BPS sensor failure, the BPS must not be spuriously activated.

<table>
<thead>
<tr>
<th>Text in requirement statement</th>
<th>Mediators</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPS is under maintenance</td>
<td>isBPSUnderMaintenace</td>
<td>Boolean</td>
</tr>
<tr>
<td>activated</td>
<td>isBPSActivated</td>
<td>Boolean</td>
</tr>
<tr>
<td>absence of any BPS component failure</td>
<td>isNoBPSComponentFailure</td>
<td>Boolean</td>
</tr>
<tr>
<td>presence of a single BPS sensor failure</td>
<td>isSingleBPSSensorFailure</td>
<td>Boolean</td>
</tr>
<tr>
<td>loss of MPS</td>
<td>isMPSLoss</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
Binging Specification Example

<table>
<thead>
<tr>
<th>Clients</th>
<th>Mediators</th>
<th>Providers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(what information is required by 1..* clients?)</td>
<td>(how to compute the required data?)</td>
</tr>
</tbody>
</table>

- isBPSUnderMaintenance
- isBPSActivated
- isNoBPSComponentFailure
- isSingleBPSSensorFailure
- isMPSLoss

System Design Alternatives or Versions

…”
Binding: What do we need to capture?

- **Mediator name** reflects what is needed by clients
- **Mediator type** must be compatible to its clients
- **Client or provider id** is the qualified name of the client or provider model (e.g. Package1.Model1.component1)
- **Client isMandatory** (default=true) indicates whether the client must be bound.

- (If needed) **Mediator template** may only contain predefined macros (e.g. sum(·), toArray(·), card(·), min(·), max(·), etc.)
- (If needed) **Client or provider template** may contain expressions including references to components within the client or provider models (e.g. in Modelica using the dot-notation)
Binging Specification Example (XML)

```xml
<mediator name="NumberOfOperatingPumps" requiredType="Integer">
  <client mandatory="true" qualifiedName="Req.numberOfOpPumps"/>
  <provider qualifiedName="PA.on">
    <template>if getPath() then 1 else 0</template>
  </provider>
  <provider qualifiedName="PB.volFlowRate">
    <template>if getPath() > 0 then 1 else 0</template>
  </provider>
  <template>sum(:)</template>
</mediator>
```

Verification Model

```plaintext
model VerificationModel1

Req req1
{
  numberOfOpPumps = sum(
    (if sys.pump1.on then 1 else 0),
    (if sys.pump2.volFlowRate > 0 then 1 else 0),
    (if sys.pump3.volFlowRate > 0 then 1 else 0)))
);

System sys(...);
Scenario scen;
end VerificationModel1;
```
Binging Specification Example

- GUI example in ModelicaML
  - Mediators group information needed by the requirement violation monitors
Proposal in MODRIO

**FORM-L**
(requirements / properties modeling language)

**Modelica**
(system behavior)

**FORM-L**
(system architecture modeling)

1. Translate to Modelica code + `bindingSpecification.xml`

2. Generate verification models

3. Simulate and analyze results
Conclusion

• The presented approach for automating model composition has the following advantages:
  – No need for modifying existing models
  – Enables a flexible integration of decoupled models (no need for fixed interfaces)
  – Exposing and grouping of information about what data is needed by clients will reduce analysis work
  – Automated generation of binding expressions will reduce modeling errors and the manual modeling effort.
  – Further application example: Traceability between models (e.g. requirements, scenario and system design alternatives)
Thank you for your attention!