An Approach for Validation of Semantic Composability in Simulation Models

Claudia Szabo and Yong Meng Teo
Department of Computer Science
National University of Singapore

email: claudias@comp.nus.edu.sg

June 23, 2009
Outline

- Introduction
- Objective
- Related Work
- CoDES Framework Overview
- Proposed Approach
- Example
- Conclusion
Composability

- “Capability to select and assemble simulation components in various combinations to satisfy user requirements” [4, 10]

- **Simulation component** - “a reusable, self contained unit that is independently testable and usable in a variety of contexts. [It] interacts with its environment only through a well defined interface of inputs and outputs.” [4]

- Various levels of composability [13]:
  - technical, **syntactic** (engineering), **semantic**, pragmatic, dynamic, conceptual ...
Validation of Semantic Composability

• Does the composed model containing the reused simulation components produce semantically correct results?

• Key issues:
  ▫ Not a closed operation [2, 4, 10]
  ▫ Emergent properties [6]
  ▫ Context [2, 13]
  ▫ Orthogonal aspects
    • Logical [5, 7]
    • Temporal [12]
    • Formal [10]
Objective

To design and develop techniques for the validation of semantic composability with formal guarantees and practical implementation potential.
Related Work

• Petty and Weisel’s formal theory [10, 11]
  ▫ Components statically represented as functions of integers; Composition = composition of mathematical functions; Validity = close enough to the simulation of a perfect model
  ▫ Static; composition is based on the linear order of components; validation relation undefined

• Validation of DEVS models [14]
  ▫ A DEVS model is represented a schema in the Z specification language; composition specification is validated using Z/EVES: inconsistencies, theorems, and syntax, type, and domain errors
  ▫ Asynchronous; assumes knowledge about entire system

• Validation of BOM models [8]
  ▫ Detailed user-specified scenario facilitates individual component discovery; candidate components executed in all possible combinations and validated against scenario
  ▫ Informal
CoDES Framework Overview

MODEL COMPOSITION

- Model Input
- Model Discovery & Selection
- Model Integration & Validation

CoDES GUI

Syntax Verifier

Model Locator (MI)

Model Selector

Semantic Validator

- Conceptual Model
- Query Request(s)
- Selected Model
- Candidate Simulator
- Validated Simulator

Model Repository

Candidate Models

Validate Communication (CI)

Concurrent process

Meta-simulation

Perfect model

June 23, 2009
CoDES Component & Simulator

- Component: abstracted as a *black-box* with *in* and/or *out* communication channel, and represented as a *meta-component*:
  - mandatory & specific attributes
  - behavior:
    - External – constraints on input and output (destination, origin, data type, range, etc.)
    - Internal – a finite timed state automaton
      
      \[
      [I_l]S_p[\Delta t] \xrightarrow{\text{Cond}_n} S_t[O_l][A_m]
      \]

- Simulator: represented by *components* (base, model) linked using *connectors*
Proposed Validation Approach

1. Evaluate composed model for desired properties
   - 1.1 Validate Component Communication
     - valid
       - Composition is valid
     - invalid
       - Composition is invalid
   - 1.2 Validate Component Coordination
     - invalid
       - Composition is invalid
     - valid
       - Composition is invalid
   - 1.3 Validate Component Coordination over Time
     - invalid
       - Composition is invalid

2. Compare composed model with perfect model
   - 2.1 Exact Match With Perfect Model
     - exact
       - Composition is valid
     - not exact
       - Composition is invalid
   - 2.2 Similar to Perfect Model
     - similar
       - Composition is valid
     - not similar
       - Composition is invalid

June 23, 2009
Layered Approach

**Validate Communication**

- Validity = Communication is correct
  - all communication is correct and meaningful

**Concurrent Process Validation**

- Validity = Safety + Liveness (for all states)
  For instantaneous transitions, for all possible interleaved executions:
  - no deadlocks
  - valid end-states
  - safety by invariance
  - liveness by progress properties

**Meta-Simulation Validation**

- Validity = Safety + Liveness (over time)
  For timed transitions, for all attributes
  - no deadlock
  - liveness for all components throughout simulation run
  - sequence of valid possible transitions

**Perfect Model Validation**

- Validity = L(M) V L(M*)
  -new time based formalism to validate model against perfect model

June 23, 2009
Concurrent Process Validation

- Validates the logical coordination of components with respect to *all* possible combinations of states

- 2 types of logical properties:
  - **Safety** – “Nothing bad ever happens” [9]
    - Entire composition: deadlock free, a property holds throughout the check
    - Individual components: a property holds throughout the check
  - **Liveness** – “Something good will always happen” [9]
    - Individual components: a specific state is always reached

- 3 Steps:
  1. Transform the composition into a specification of concurrent processes
  2. Specify safety and liveness
  3. Validate using a model checker
mtype {Job}; chan to1 = [10] of {mtype}; chan to2 = [10] of {mtype}; ...

hidden byte sourceIAMax = 10; byte sourceIATime; byte noJobsSource = 0;

proctype CON_ONE_TO_ONE(chan in, out){
do :: in ? Job -> out ! Job; od
}

proctype SOURCE1(int id, noJobsMax; chan out){
do :: (sourceIATime == sourceIAMax) -> sourceIATime =0;
if :: out ! Job -> progress: printf("[Source] Job sent
"); noJobsSource ++; fiod }

proctype SourceCounter(){
do :: (sourceIATime < sourceIAMax) -> sourceIATime++; od
}

proctype SINK1(int id; chan in){
S1: atomic{
if :: in ? Job -> printf("[Sink] Job received!
"); goto S1; fi})

proctype SERVER3(int id; chan in, out){
bit busy;
S1: {
if :: in ? Job -> printf("[Server] Job received!
"); busy=1;goto S2; fi
S2: {
if :: out ! Job -> progress: printf("[Server] Job sent! 
"); busy=0; goto S1;}}
active proctype monitor(){assert (noJobsSource < noJobsMax);}
Meta-Simulation Validation

• Guarantees composition safety and liveness through time

• 3 Steps:
  1. Transform composition into Java classes considering components’ state machines through time
  2. Specify desired logical properties to be validated through time
     • Safety – Validity points - data that must pass through various connection points in the composition
       • VP1 = d1 {origin = Server, destination = Sink, range = 10; 35, type = double}
     • Liveness – Transient predicate - predicate that must become false during a given time interval after it becomes true.
       • transient(Server) = (busy == true)
  3. Validate by running the meta-simulation and observing the logical properties
Perfect Model Validation

- Proposes formal validation of model execution through time

5 Steps:
1. [Formal Component Representation] Represent a component as a function of states over time
2. [Unfolding and Sampling] Transform formal component representation using sampled time values
3. [Composition] Compose functions mathematically
4. [Simulation] Run the functions as a simulation and obtain LTS
5. [Validation] Compare the composed model with a perfect model
Perfect Model Validation (2)

1. [Formal Component Representation] Represent each component and perfect component as a function of states over time.

   \[ f_i : I_i \times S_i \times T_i \rightarrow O_i \times S_i \times T_i \]

   \[ f_i^* : I_i^* \times S_i^* \times T_i^* \rightarrow O_i^* \times S_i^* \times T_i^* \]

1. [Unfolding and Sampling] Sampling is done using the attribute values provided by the user.

2. [Composition] Verify that \( f_j \) are composable using domain and co-domain inclusion. Same for \( f_i^* \).

   - Composability:

Given \( f_i \) and \( f_j \) that describe adjacent components, with \( f_j \) requiring input from \( f_i \). \( f_i \) and \( f_j \) are composable and we can write \( f_j(f_i) \) if each attribution of time moments for each unfolding of \( f_j \) when \( f_j \) requires input is greater or equal to the attribution for \( f_i \) when \( f_i \) produces output.
Perfect Model Validation (3)

4. [Simulation] Represent each simulation run as an LTS, $L(M)$ and $L(M *)$ where

- **nodes**: $\bigcup_{s \in S}$ where $S$ is the state of all components in the system
- **edges**: function calls $f_i$ and $f_i^*$ in the simulation run
- **labels**: tuple $\langle \text{function\_name, duration, output} \rangle$

4. [Validation]
   a. Determine **strong** bisimulation relations between $L(M)$ and $L(M *)$
   b. If no strong bisimulation relation exist, determine whether $V_\varepsilon$ is a **weak** bisimulation relation
Petty & Weisel’s Formal Theory

1. Formal component representation

- *static*
- *unfeasible*

1. Composition

- *component order*

\[ M = f_3 \circ f_2 \circ f_1 \]

1. Validation

- *less meaningful*
- *strong*
- *semantically close*:

\[ V \in \varepsilon \]

Our approach

- *time-based*
- *dynamic*

- *execution order wrt time*
Example

1. + 2. [Formal Component Representation] + [Unfolding and Sampling]

\[ f_1 : \emptyset \times S_1 \times T_1 \rightarrow \{O_1\} \times S_1 \times T_1 \]

\[ f_1(\emptyset, s_i, t) \rightarrow (O_1, s_i', t + \Delta t) \]

Unfold for \( \tau = 3 \)

Sample with mean = 4 : \( \Delta t = 6, \Delta t = 2, \Delta t = 4 \)

<table>
<thead>
<tr>
<th>Unfold</th>
<th>( \Delta t )</th>
<th>Formula</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \frac{1}{2} )</td>
<td>( f_1(0, s_1, 0) )</td>
<td>( f_1(0, s_1', 6) )</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>( f_2(0, s_2, 6) )</td>
<td>( f_1(0, s_1', 8) )</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>( f_3(0, s_3, 8) )</td>
<td>( f_1(0, s_1', 12) )</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>( f_2(2, x_2, x_2', 0) )</td>
<td>( f_1(2, x_1', 11) )</td>
</tr>
<tr>
<td>2</td>
<td>( \frac{3}{2} )</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>( f_2(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>( f_3(2, x_2^1, x_2^1, 6) )</td>
<td>( f_1(2, x_1', 6) )</td>
</tr>
</tbody>
</table>
Example (2)

3. [Composition] The most trivial constraints that can be defined are:

\[ x \geq \Delta w_1, t \geq x + 11, t \geq 8 + \Delta w_2, r \geq t + 6, r \geq 12 + \Delta w_3 \]

\[ x' \geq x + 11 + \Delta w_1', t' \geq x' + 1, t' \geq t + 6 + \Delta w_2', r' \geq t' + 1, r' \geq r + 1 + \Delta w_3' \]

\[ (x = 8, t = 19, r = 25), (x' = 23, t' = 28, r' = 29). \]

\[ (x^* = 8, t^* = 19, r^* = 25), (x'^* = 23, t'^* = 28, r'^* = 29). \]
Example (3)

4. [Simulation]

\[
\begin{align*}
&f_1(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^2, 6) \\
&f_1(\emptyset, s_1^1, 6) \rightarrow (O_1, s_2^2, 8) \\
&f_2(I_2, s_2^2, 8) \rightarrow (O_2, s_2^2, 19) \\
&f_1(\emptyset, s_1^1, 8) \rightarrow (O_1, s_4^4, 12) \\
&f_2(I_2, s_2^2, 19) \rightarrow (O_2, s_4^4, 25) \\
&f_3(I_3, s_3^3, 23) \rightarrow (\emptyset, s_2^2, 24) \\
&f_2(I_2, s_2^2, 25) \rightarrow (O_2, s_4^4, 26) \\
&f_3(I_3, s_3^3, 28) \rightarrow (\emptyset, s_3^3, 29) \\
&f_3(I_3, s_3^3, 29) \rightarrow (\emptyset, s_4^4, 30)
\end{align*}
\]

Composition

\[
\begin{align*}
&f_1^0(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^2, 6) \\
&f_1^0(\emptyset, s_1^1, 6) \rightarrow (O_1, s_2^2, 8) \\
&f_2^0(I_2, s_2^2, 8) \rightarrow (O_2, s_2^2, 19) \\
&f_1^0(\emptyset, s_1^1, 8) \rightarrow (O_1, s_4^4, 12) \\
&f_2^0(I_2, s_2^2, 19) \rightarrow (O_2, s_4^4, 25) \\
&f_3^0(I_3, s_3^3, 23) \rightarrow (\emptyset, s_2^2, 24) \\
&f_2^0(I_2, s_2^2, 25) \rightarrow (O_2, s_4^4, 26) \\
&f_3^0(I_3, s_3^3, 28) \rightarrow (\emptyset, s_3^3, 29) \\
&f_3^0(I_3, s_3^3, 29) \rightarrow (\emptyset, s_4^4, 30)
\end{align*}
\]

Perfect Composition

June 23, 2009
5. **[Validation]**

1. Strong bisimulation relation between $M$ and $M^*$ validated by the CADP toolset [32]
Implementation

- Validation of composed model (Concurrent Process Validation)
  - Java program transforms the COML files into a Promela specification
  - The Spin model checker is called to validate the Promela specification

- Validation of composed model through time (Meta-Simulation Validation) (Java)
  - Java program transforms the COML files into a Java hierarchy
  - Threaded implementation using PipedInputStreams

- Validation against perfect model (Perfect Model Validation)
  - Java program transforms component COML files into functional representation and subsequently into LTS; uses the BISIMULATOR tool from the CADP toolset to determine strong equivalence
  - Java program determines related states according to proposed semantic metric relation; related states are subsequently validated by BISIMULATOR
Weaknesses

- Validation of composed model
  - State explosion
    - 3 components with 2 states/comp – 2 minutes
    - 10 components with 2 states/comp – 15 minutes
  - Can be limited by further abstracting the Promela representation – accuracy vs complexity

- Validation of composed model *through time*
  - Based on sampling - #samples vs computation cost

- Validation against perfect model
  - Based on sampling - #samples vs computation cost
  - Semantic metric relation – formal measure vs semantic meaning of validity
Conclusion

• Three-layer approach for semantic validation of compositions with increasing accuracy and complexity:
  ▫ Validation of composed model:
    1. Formal guarantee of the logical component coordination
    2. Safety and liveness over time
  ▫ Validation against perfect model:
    3. Formal guarantee of composition validation through time

• Implementation tested on simple and complex examples of open queueing networks
Thank you! Questions?


Semantic Parametric Metric Relation

\[ p = [s(p), f_{in}(p), f_{out}(p)] \]
\[ q = [s^*(q), f^*_{in}(p), f^*_{out}(p)] \]

\[ V_\varepsilon(p,q) = \{(p,q) \in P \times Q \mid \|p - q\|_\sigma \leq \varepsilon \} \]

\[ \|p - q\|_\sigma = DS(s(p), s^*(q)) + \frac{DF(f_{in}(p), f^*_{in}(q)) + DF(f_{out}(p), f^*_{out}(q))}{2} \]

**determine if composition states are related (attributes and values)**

**determine if the same component is executed**