CIMSIM2012 Keynote

Component-based Modeling and Simulation

Yong Meng TEO
Department of Computer Science
National University of Singapore
email: teoym@comp.nus.edu.sg
Motivation

• **Model Development and Simulation**
  - *program centric, model-centric* to **model sharing** [31]
  - reuse of simulation models is appealing
    - reduced development time and cost [31]
    - flexibility in meeting user requirements [65]

• **Simulation Operating Environment**
  - Internet as infrastructure for model sharing – grid, peer-to-peer, cloud computing, etc.

• **Knowledge Sharing**
  - models (knowledge) sharing on a larger scale
  - quantum leap in size and capability of simulation applications
Outline

• Motivation

• Related Work

• Component-based Modeling & Simulation
  ▪ Modeling & Simulation Life-cycle
  ▪ Cross-cutting Issues
  ▪ Syntactic Composability Verification
  ▪ Semantic Composability Validation

• Summary

• References
Composability

“capability to select and assemble simulation components in various combinations to satisfy user requirements” [94]

• Simulation component - “a reusable, self contained unit that is independently testable and usable in a variety of contexts.” [31]
Levels of Composability

- Conceptual
- Dynamic
- Pragmatic
- Semantic
- Syntactic
- Technical
- None

25 September 2012
CIMSIM 2012 - Keynote
Vision

COMPONENT TOOLBOX

Queueing Networks

reuse of base components

Simulation Model for a Cluster Scheduler

reuse of model components

Simulation Model for a Grid Meta-scheduler
Component-based Approaches

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Representation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>black-box</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>stateful</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>time</td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Composition</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hierarchical components</td>
<td>-</td>
<td>-</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>component-connector paradigm</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>distributed components</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Framework</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>general purpose</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>component marketplace</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Representation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>black-box</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>stateful</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Composition</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hierarchical components</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>component-connector paradigm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>distributed components</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Framework</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>general-purpose</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>component marketplace</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Software Engineering vs Modeling & Simulation

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Component-based Software Engineering</th>
<th>Component-based Modeling and Simulation</th>
</tr>
</thead>
</table>
| **Component Representation** | ● describes method syntax [123]  
● stateless [5, 111, 115]  
● no time [111, 115, 123]  
● simple components [123] | ● describes component behavior [132]  
● stateful [30, 61, 75, 88]  
● time based [61, 12, 132]  
● complex models [30] |
| **Composition** | 1. Syntactic Composability  
 ● runtime environment for component connection and composition execution [2, 5, 103] | runtime environment for component connection and composition execution [22, 28, 29, 133] |
|                     | 2. Domain Knowledge  
 ● ontology used for discovery [123] | ontology used to translate application domain concepts to simulation concepts [109] |
|                     | 3. Semantic Composability  
 ● not considered [2, 5, 103, 63] | desired [12, 61, 88] |
|                     | 4. Validation  
 ● program correctness [123, 68] | program correctness + closeness of composed model to real system [10, 88] |
| **Framework** | ● integrated framework desired [123] | ● integrated framework desired [61, 12] |
Related Work

• Cross-cutting issues [12, 40]
  ▪ Component abstraction [32, 40]
  ▪ Model reuse [12, 32, 40, 87, 96, 101, 135]
  ▪ Model discovery and selection [45, 92, 98, 101, 135]
  ▪ Syntactic verification [23, 54]
  ▪ …

• Semantic validation is non-trivial [95, 98]
  ▪ Not a closed operation
  ▪ Many aspects: data compatibility, logical, temporal, formal, …
  ▪ Emergent properties
  ▪ ….
Outline

• Motivation

• Related Work

• Component-based Modeling & Simulation
  ▪ Modeling & Simulation Life-cycle
  ▪ Cross-cutting Issues
  ▪ Syntactic Composability Verification
  ▪ Semantic Composability Validation

• Summary

• References
Life-cycle of Simulation Study

Traditional (adapted from Banks et al., 2005)

Proposed Component-based Life-cycle
Cross-cutting Issues in Component-based Modeling & Simulation

Syntactic Verification

Model Discovery

Model Selection

Semantic Validation

- component representation
- model reuse
- data encapsulation
- loose coupling
- component repository
- heterogeneous components
- domain knowledge
CoDES: Composable Discrete-event Scalable Simulation

[ANSS2008]
Outline

• Motivation
• Related Work

• Component-based Modeling & Simulation
  ▪ Modeling & Simulation Life-cycle
  ▪ Cross-cutting Issues
  ▪ **Syntactic Composability Verification**
    ▪ Model and Component Representation
    ▪ Hierarchical Component Reuse
    ▪ Describing Domain Knowledge
    ▪ EBNF Composition Grammar
    ▪ Example: Queuing Networks
    ▪ Implementation
  ▪ Semantic Composability Validation

• Summary
• References

Syntactic Composability Verification

• “capability to combine and recombine components without substantial integration efforts” [3]

• How (a formalism to)
  ▪ Guides the composition
  ▪ Represents conceptual model for discovery & validation
Syntactic Verification Process
Model Representation

- A network of **components** linked by **connectors**

- Types of connectors:
  - One to One
  - One to Many (Fork)
  - Many to One (Join)

- Component-Connector Paradigm + components as black-boxes = *loose coupling* and *data encapsulation*
Component Representation

• A **black-box** with *in* and/or *out* communication channels

• *Meta-component* abstraction:
  ▪ attributes: mandatory & specific
  ▪ behavior: a finite timed state automaton

\[
[I_l]S_p[\Delta t] \xrightarrow{\text{Cond}_n} S_t[O_l][A_m]
\]

• Implemented using our proposed COML standard (XSD schema)
Hierarchical Component Reuse

within a domain

across domains
Describing Domain Knowledge

• Enrich model description to support syntactic verification, discovery, semantic validation, …

• **COSMO** (*CO*mmponent *Sim*ulation and *Mod*eling *Ont*ology)
  - component-oriented simulation within and across application domains
  - high-level ontology (simulation); low-level ontology (application domain): OWL-DL and RDF
COSMO Ontology: Queuing Network Example
EBNF Composition Grammar

• 2 parts:

(a) CoDES Composition Rules

Simulator := (Comp Con)+
Con := ConO | ConF | ConJ
Comp := Base_Comp | Model_Comp
Base_Comp := QN_B_Comp

#CoDES Reuse Rule
Model_Comp := QN_Simulator | MC_Simulator

✓ Formal
✓ Reuse across app domains
✓ Scalable

Example: Queuing Networks

(b) Domain Specific Composition Rules

#Base Components
QN_B_Comp := Source|Server|Sink

#QN Composition Rules
QN_Simulator := Source BlockNT+ [Terminal]|Source BlockT+
Terminal := ConO End|ConF ("("End")")|ConJ End
End := Source|Sink

BlockNT := ConF ("("Box BlockNT* [ConJ Box BlockNT*]"")")|ConO Box [BlockNT]+|_

BlockT := ConF ("("Box BlockT* [Terminal|ConJ BlockT*]"")")|ConO Box [BlockT]+|_

Box := Server|Model_Comp
Implementation Details

- **Component representation**
  - COML schema implemented in XSD and XML
  - validated using XML parser
- **COSMO ontology**
  - created using Protégé
  - specified using RDF (Resource Description Framework)
- **Composition grammar**
  - EBNF
  - Java parser using Earley’s algorithm
- **Model discovery & selection**
  - implemented in Java
  - Matching Index computation uses Jena Reasoner on the COSMO ontology
- **Verification of syntactic composability**
  - composition grammar parser written in Java
Outline

• Motivation
• Related Work

• Component-based Modeling & Simulation
  ▪ Modeling & Simulation Life-cycle
  ▪ Cross-cutting Issues
  ▪ Syntactic Composability Verification
    ▪ Approach
    ▪ General Model Properties
    ▪ Formal Model Execution
    ▪ Validation Cost

• Summary
• References
Semantic Composability Validation

• Does a composed model produces semantically correct results?

• Key issues:
  ▪ Not a closed operation [10]
  ▪ Various perspectives [88]:
    ▫ Logical
    ▫ Temporal
    ▫ Formal
    ▫ Accuracy
    ▫ Cost
    ▪ …
Approach

• Semantic validation is computationally expensive

• Observations
  ▪ More invalid than valid models
  ▪ Discarding a model as invalid is generally cheaper than proving absolute validity

• Incremental approach
  ▪ Progressively eliminates invalid models
    □ Increases accuracy and credibility
    □ Higher computational cost
Model Population
Approach – **Syntactic Verification**

- **VALID** (syntax ✓)
- **INVALID** (syntax ✗)
Approach – Semantic Validation

Invalid (syntax ✗, semantic ✗)

Invalid (syntax ✓, semantic ✗)
Approach – Semantic Validation

Valid = Syntax + Semantic (model properties + model execution)
1. General model properties
   ✓ Correct and meaningful component communication
   ✓ Safety, liveness, deadlock-free – with time
     - Covers software engineering + simulation perspectives
     [PADS 2009, PADS 2010]

2. Formal validation of model execution
   ✓ Comparison with reference model using state machine
     - New time-based formalism defines validity
     - 5-step process validates model execution over time
     [WinterSim 2009, PADS 2011]
Layered Approach Detail

1. Model Properties
   - communication
   - safety, liveness

2. Model Execution
   - compare composed model execution with reference model

1.1 Validate Component Communication

1.2 Validate Component Coordination

1.3 Validate Component Coordination over Time

2.1 Exact Match with Reference Model

2.2 Similar with Reference Model

Concurrent Process Validation

Meta-Simulation Validation

Time-based Formal Validation

increasing accuracy and costs
1.1 Component Communication

- Quantifies degree of *semantic compatibility* between the reused components with respect to newly composed model

- Composability Index:

\[
\text{Comp} = \{(C_i, C_j) | C_i = \langle R_{C_i}, A_{C_i}, B_{C_i} \rangle, C_j = \langle R_{C_j}, A_{C_j}, B_{C_j} \rangle\}
\]

\[
CI(\text{Comp}) = \frac{\sum SAT(ic, OC_{C_i})}{\sum |IC_{C_j}|}
\]

\[
SAT(ic, OC_{C_i}) = \begin{cases} 
1 & \exists oc \in OC_{C_i} \text{ such that type}(oc) = \text{type}(ic), \\
0 & \text{otherwise}
\end{cases}
\]
1.2 Component Coordination

- Guarantee that the logical coordination of components is correct
- Validate the logical coordination of components with respect to all possible combinations of states; transitions are instantaneous

- Properties:
  - Safety – “Nothing bad ever happens”
    - Entire composition: deadlock free, a property holds throughout the check
    - Individual components: a property holds throughout the check
  - Liveness – “Something good will always happen”
    - Individual components: a specific state is always reached

- Represent using Promela + employ SPIN model checker
- Suffers from state explosion => attributes and time are abstracted away
1.3 Meta-simulation Validation

• Guarantees composition safety and liveness through time
• We propose validity points to specify safety and transient predicates to specify liveness

• Properties
  □ Safety – Validity points [PADS2009] - 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(C2) = (busy == true)

• Transform into Java hierarchy and execute
2 Formal Model Execution Validation

• Software engineering
  ▪ Model properties - safety, liveness, deadlock-free, …
  ▪ Simple state machines and few attributes; often timeless

• Simulation
  ▪ Complex state machines & larger #attributes that change over time => insufficient
  ▪ Composed model is a valid abstraction of real system => requires validation of model execution over time
  ▪ Current state-of-the-art (Petty&Weisel) propose a timeless formalism => general model properties are previously validated; no fork/join; no feedback loops
Comparison with Petty & Weisel’s Formal Composability Theory

1. Formal component representation

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

- static
- unfeasible

2. Composition

- component order
- no fork & join
- no feedback loops

3. Reference model

- assume to exist a-priori

Our approach

- time-based
- dynamic

- time-based
- fork & join
- feedback loops

constructed from generic description

25 September 2012
Time-based Formalism

• Component
  ▪ function representing input, states & time vs integer function

\[ f_i : X_i \rightarrow Y_i \]
\[ X_i = I_i \times S_i \times T_i, Y_i = I_i \times S_i \times T_i \]

• Mathematical composability
  ▪ time vs integer domains

Expressive

Given a composed model \( M = \{(f_i, f_j) | i, j = 1, n\} \) then \( f_i \) and \( f_j \) are composable if there exists the bijective binary relation \( R = \{(t^{(j)}_n, t^{(i)}_m) \in T^{in}_j \times T^{out}_i | t^{(j)}_n > t^{(j)}_m\} \)

Orders time
Time-based Formalism (2)

- **Simulation**
  - Time-ordered functions => fork, join, feedback loops

  \[ S(M) = \{ [\ldots f_i(I_p^i, S_p^i, t_p^i) \rightarrow (O_p^i, S_{p+1}^i, t_{p+1}^i), \ldots, f_j(I_q^j, S_q^j, t_q^j) \rightarrow (O_q^j, S_{q+1}^j, t_{q+1}^j) \ldots ] \} \]

- **Validity**: compares with reference model over time
  - Exact match
  - New degree of similarity index based on related simulation states \( V_\varepsilon \)

\[ \text{Measure of similarity} \]
Degree of Semantic Similarity \( (V_\varepsilon) \)

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

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

- composition states are related:
- related attributes (in the ontology)
  - same value or
  - same trend
- same component is executed
5-Step Validation Process

Composed Model

1. Formal Representation
   \[ C_1 = f_1 \]

2. Unfolding and Sampling

3. Composition

4. Simulation

Reference Model

1. Formal Representation
   \[ C_1^* = f_1^* \]

2. Unfolding and Sampling

3. Composition

4. Simulation

Strong equivalence?

L(M) \( \subseteq \) L(M*)?

If Yes, then Valid

If No, then L(M) \( \not\subseteq \) L(M*)

Invalid
Validation Cost

- Simple example
  - #components = 1,000
  - fork/join = 10%
  - #states = 3 states/component
  - no feedback loops

- Larger example
  - #components = 20
  - fork/join = 25%
  - #states = 9 states/component
  - feedback loops = 10%

More results in [PADS2011, JoS2012]
Simple Example

General Model Properties - Formal Validation
5 - 55%

*It takes 49 seconds to simulate and 17 minutes to validate*

Large Example

General Model Properties - Formal Validation
87 – 13%

*It takes 1.3 minutes to simulate and 27 minutes to validate*
Summary

• **Component-based modeling**
  - **Component abstractions**: meta-components, domain knowledge representation [ANSS 2008]
  - **Model discovery and selection**: partial matches [ANSS 2008, WinterSim2011]

• **Syntactic composability verification**
  - **EBNF composition grammar** [AMS 2007, PADS2010]

• **Semantic composability validation**:
  - **Approach** [PADS 2009, PADS 2011]
  - **General model properties** [PADS2009, PADS2010]
  - **Formal time-based formalism of model execution** [Wintersim 2009*]
  - **Costs of validation** [PADS2011, JoS2012]

* ACM SIGSIM Best PhD Student Paper Award
Summary

Simulation examples: queuing networks, grid virtual organizations, military troop training
Open Problems

1. Who (and how to specify) the reference model for validation?

2. Techniques to reduce model state space in larger models

3. What about emergent properties?
Emergent Properties

“The whole is greater than the sum of its parts” - Aristotle

System Properties

Design Properties

Emergent Properties

Syntactic Verification
Semantic Validation

Prediction or Detection
Semantic Validation

Publications


Acknowledgement: This keynote is based on the PhD work my student, Dr Claudia Szabo, currently with the University of Adelaide, Australia.