On Syntactic Composability and Model Reuse

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

**Asia-Pacific Science & Technology Centre
Sun Microsystems Inc.
email: teoym@comp.nus.edu.sg

Outline

- Motivation
- Objective
- Related Work
- Design of CODES
  (Component-based Discrete-Event Simulation)
  - Definitions
  - Composition Grammar
  - Example - Queuing Network, Model Composition, Model Reuse
- Prototype Implementation
- Summary
Motivation

- Development of Modeling and Simulation
  - program-centric $\rightarrow$ model-centric $\rightarrow$ model sharing
  - 1-1 $\rightarrow$ 1-many $\rightarrow$ many-to-many
  - simulation models reuse to reduce simulation development time and cost

- Simulation Environment
  - increasing trend to use Internet-based infrastructure for more pervasive sharing of resources – grid, p2p, web services, service-oriented architecture, etc.
  - advance simulation models and knowledge sharing on a larger and wider scale
  - quantum leap in capability and the size of the simulation applications

Objective

To design and develop methodology, abstraction and techniques to facilitate service-oriented simulation model sharing through *composable models.*
Some Issues

- **Model discovery**
  - NP complete problem
  - Dependant on component representation
  - Expose components as services? (Service Oriented Modeling and Simulation)

- **Syntactic validity** – component interoperability

- **Semantic validity** – is the composition semantically meaningful?

- **Reuse**

- **Model (Component) representation**? Attributes, functionality, constraints, ….

---

Related Work

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Syntax</th>
<th>Semantic</th>
<th>Reuse</th>
<th>Discovery</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Box</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>COST (02)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>JSIM (99)</td>
</tr>
<tr>
<td>DEVS (Discrete Event System Specification)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>DEVSSJava (00)</td>
</tr>
<tr>
<td>BOM (Base Object Model)</td>
<td>✓</td>
<td>✓?</td>
<td>✓</td>
<td>✓?</td>
<td>BOMWorks (04?)</td>
</tr>
<tr>
<td>Fractal Component Model</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OSA (06)</td>
</tr>
</tbody>
</table>
Design of CODES
(Component-based Discrete-Event Simulation)

- **layered approach** to facilitate model discovery - syntactic then semantic

- A CODES simulator consists of **components** linked by **connectors**.

- A **component** is a black-box with an input communication channel (‘in’) and an output communication channel (‘out’).

- A **connector** links two or more components and is one of three types: one-to-one, one-to-many (fork) and many-to-one (merge).
Design of CODES (2)

- Hierarchical component architecture to support model composition and reuse within and across application domains

![Diagram]

Design of CODES (3)

- Extendable syntactic composition rules → composition grammar defines composition and reuse rules

- CODES EBNF Composition Grammar – divided into 2 parts

(a) **Base Composition Rules**

```
# CODES Composition Rules
Simulator ::= (Comp Con)+
Con ::= ConO | ConF | ConJ
Comp ::= B_Comp | Rep_Comp
B_Comp ::= QN.B_Comp
```

extend by adding rules for new application domains

```
# CODES Reuse Rule
Rep_Comp ::= QN.R_Simulator | QN.E_Simulator
```
Design of CODES (4)

(b) Application specific composition rules

# Application Specific

# Base Components

QN_B_Comp ::= Source | Server | Sink

# QN Composition Rules

QN_Simulator ::= Source BlockNT+ Terminal?
   | Source BlockT+
Terminal ::= ConO Final | ConF ("(" Final ")")+
   | ConJ Final
Final ::= Source | Sink
BlockNT ::= ConF ("(" BlockBox BlockNT*)
   (ConI BlockBox BlockNT*)
   | (ConO BlockBox BlockNT*)+
   | _
BlockT ::= ConF ("(" BlockBox BlockT*)
   (Terminal | ConI BlockBox BlockT*)
   | (ConO BlockBox BlockT*)+
   | _
BlackBox ::= Server | Rep_Comp

Queuing Networks – Components & Connectors
Example of Model Composition

- Number of Components: 11
- Production String:
  \[
  \text{Grid}_\text{QN} = \text{Source} \con O \text{ Server} \con F \text{ (Server ConJ Sink) (Server ConJ Sink)} \text{ (Server ConJ Sink) (Server ConJ Sink) (Server ConF (Server ConJ Sink)(Server ConJ Sink))}
  \]

A Computational Grid with Two Virtual Organizations

Example of Model Reuse

- Assume VOs are models found in the model repository
  - Virtual Organization 1:
    \[
    \text{VQ1}_\text{QN} = \text{Server ConF (Server ConJ)(Server ConJ) (Server ConJ)}
    \]
  - Virtual Organization 2:
    \[
    \text{VQ2}_\text{QN} = \text{Server ConF (Server ConJ)(Server ConJ)}
    \]

- Reusing model components

- Number of Components: 6
  \[
  \text{Grid}_\text{QN} = \text{Source} \con O \text{ Server ConF (Rep_Comp Sink)} \text{ (Rep_Comp Sink)}
  \]
Prototype Implementation

- Purpose – experimental test-bed for syntactic composability
- Base components implemented using SSF (Scalable Simulation Framework) Java API
- CODES GUI
  - Simulation Modeling
    - Graphical input of simulation model
    - Syntactic composition checks
  - Simulation Execution
    - Automatic code generation – Java-based simulator using SSF
- Demo

Summary

- focus on syntactic composability
  - use of EBNF and hierarchical component definitions to describe/represent syntactic composability
  - support component (model) discovery and reuse across application domains
  - extendable to support new application domains
- further work
  - composability index to quantify degree of syntactic match of models - facilitate model discovery and reuse
  - Semantic composability

My webpage: www.comp.nus.edu.sg/~teoym