Systems and Software engineering with/out Simulation: State of the Art and Way Forward

Sept 25, 2019

Bernard P. Zeigler
Prof. Emeritus U. Arizona
Chief Scientist, RTSync Corp.
IEEE 2019-2021 Distinguished Speaker
Outline

• Modeling and Simulation (M&S) represents a core capability needed to address today’s complex, adaptive, systems of systems (SoS) engineering challenges.

• The limitations of Model-Based Systems Engineering (MBSE) include limited capability to develop multifaceted models, as well as their analysis with computationally powerful and correct simulation engines.

• Software engineering has become a primary implementation for SoS development so it must also be brought into the discussion.

• We discuss potential for closer integration between the three streams.
Empowering the people who drive technology.
Who We Are

The IEEE Computer is the world’s home for computer science, engineering, and technology. Known as the organization that empowers the people who drive technology, the IEEE Computer Society brings forward-thinking technology professionals together to discover the next technological innovation, develop international standards, form communities, and share knowledge.

Our Vision:

To be the leading provider of technical information, community services, and personalized services to the world’s computing professionals.
IEEE Computer Society Today

Global Reach

- 60,000+ Members
- 480 Chapters
- 168 Countries

Technical Breadth

- 200 Conferences
- 600,000+ Trusted Technical Documents
- 30+ Top-cited Periodicals
- 121+ Active Standards
Student Scholarships
$40,000+ Available Each Year

Richard E. Merwin Student Ambassador
• 36, $1000 Scholarships Awarded
• Deadlines: April 30 & September 30
• Designed to recognize and reward active student volunteer leaders in student branches or student branch chapters

Lance Stafford Larson Paper Contest
• 2, $500 Scholarships Awarded
• Deadline October 31

Upsilon Pi Epsilon Honor Society
• 4, $1,000 Scholarships Awarded
• Deadline: October 31
Student Scholarships

$40,000+ Available Each Year

Distinguished Visitors Program

• Provides professional and student chapters top technology leaders and innovators from every sector of the computing industry.
• New speakers and new topics are added on an ongoing basis.
Modeling and Simulation in support of Systems and Software Engineering

Systems of Systems (SoS) Engineering

MBSE

Software Engineering

Architectural Design

Modeling and Simulation

Test and evaluation, Conflict management, V&V
Modeling and Simulation in support of Systems and Software Engineering

- Systems of Systems (SoS) Engineering
- MBSE
  - SysML

- Software Engineering
  - Architectural Design
    - UML
    - DEVS
    - SoSADL

- Modeling and Simulation
  - Test and evaluation, Conflict management, V&V
  - Smart cities, AI-based technology
  - Cyber-physical systems
  - Internet of Things

- Internet of Things
- Systems of Systems (SoS) Engineering
History of MBSE and DEVS

A. Wayne Wymore

B.P. Zeigler
Theory of Modeling and Simulation, 1976
Discrete Event System Specification (DEVS)

Grady Booch, et al
Unified Modeling Language (UML), 1995

Systems Modeling Language (SysML), 2001

Model-Based Systems Engineering, 1993

Multi-formalism Modeling

?-
# Theory of Modeling and Simulation

## Levels of System Specification

<table>
<thead>
<tr>
<th>Level</th>
<th>Specification Name</th>
<th>What we know at this level</th>
<th>Example: A Person in a Conversation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Observation Frame</td>
<td>How to stimulate the system with inputs; what variables to measure and how to observe them over a time base;</td>
<td>The person has inputs and outputs at the usual cognitive level, such as streams of words</td>
</tr>
<tr>
<td>1</td>
<td>I/O Behavior</td>
<td>Time-indexed data collected from a source system; consists of input/output pairs</td>
<td>For each input that the person recognizes, the set of possible outputs that the person can produce</td>
</tr>
<tr>
<td>2</td>
<td>I/O Function</td>
<td>Knowledge of initial state; given an initial state, every input stimulus produces a unique output.</td>
<td>Assuming knowledge of the person's initial state when starting the conversation, the unique output response to each input.</td>
</tr>
<tr>
<td>3</td>
<td>State Transition</td>
<td>How states are affected by inputs; given a state and an input what is the state after the input stimulus is over; what output event is generated by a state.</td>
<td>How the person transits from state to state under input words and generates output words from the current state.</td>
</tr>
<tr>
<td>4</td>
<td>Coupled Component</td>
<td>Components and how they are coupled together. The components can be specified at lower levels or can even be structure systems themselves – leading to hierarchical structure.</td>
<td>A description of a person's I/O behavior in terms of neural components and their interaction by spikes is at this level.</td>
</tr>
</tbody>
</table>

## Basic Entities in M&S

<table>
<thead>
<tr>
<th>Basic Entity</th>
<th>Definition</th>
<th>Example: A Person in a Conversation</th>
</tr>
</thead>
<tbody>
<tr>
<td>source system</td>
<td>real or artificial source of data</td>
<td>Participants’ in a conversation</td>
</tr>
<tr>
<td>behavior database</td>
<td>collection of gathered data</td>
<td>I/O Behavior as in levels of system specification</td>
</tr>
<tr>
<td>experimental frame</td>
<td>specifies the conditions under which system is observed or experimented with</td>
<td>Observation of participants’ stream of words in a conversation</td>
</tr>
<tr>
<td>model</td>
<td>instructions for generating data</td>
<td>Coupled model of Finite state generator and recognizer implemented in neural form</td>
</tr>
<tr>
<td>simulator</td>
<td>computational device for generating behavior of the model</td>
<td>Discrete Event Simulation Environment</td>
</tr>
</tbody>
</table>
Equivalences at each Level of System Specification
DEVS Model of Spiking Neural Net

Atomic Model: Leaky Integrate and Fire (LIF) Neuron

Coupled Model: Neural Net

Internal couplings:

\[ I_d \text{ is the set of influencers of } d, I_d \subseteq D, d \notin I_d \]

\[ Z_j \otimes \quad i \in I \quad Y_i \rightarrow X_j \]

\[ Z_j(...) = F(<w_i,y_i>) \]
A behavior required by self-organizing NN
A behavior required by self-organizing NN

Left and Right cells have identical internal structure...
you are able to identify who receive the first and second inputs...
using signals and states

https://www.youtube.com/watch?v=UPCAqHP9GYo
A seemingly simple behavior that spiking LIF neuron model can’t display – we need suitably configured higher level unit to implement it.

\[ \delta_{\text{ext}}(s,p) \]

\[ y_i \]

One neuron per state

S2 neuron persists in state

S2 neuron activates S3 neuron

x and y are coded as spike and cannot take s0 to different non-equivalent states as required.
DEVS Systems-based Simulation is applicable to wide spectrum complex systems

• Theoretical – supporting application
  – Closure under coupling, universality, uniqueness, relation to other formalisms
  – Hierarchical Model Construction supports complex system development
  – Supports the correctness of the algorithms and validation of the executing models, e.g., time management is rigorously defined
  – Discrete Event System Specification (DEVS) formalism can be easily expanded beyond discrete-event world to continuous characteristics of system (Hybrid-DEVS)

• In Application
  – Models, Simulators and Experimental Frames are distinct entities with their own software representations
  – Precise and well-defined mathematical representation
  – Models/Experiments are developed systematically for interoperability
  – Repositories of models/experiments created and maintained systematically
  – Components can be easily reused for constructing new models
  – Discrete-event basis improves performance (e.g. no need for a global clock to control timing)
  – DEVS software can be deployed on distributed computing environments and interact with heterogeneous M&S system.
Application Example – European IOT Software Project

Approach

Secure and context-aware orchestration of sensors, actuators and software services
Language to specify Devices behavior
Risk-Driven Decision Support
Automated deployment of Smart IoT systems
Dynamic adaptation in open contexts & actuation conflicts handling

Simulation, Emulation, and Test environment for Smart IoT applications.
Actuation conflict identification and management

Run-time quality assurance and root-cause analysis
Actuation Conflict Management in large scale

- Application before Actuation Conflict Management

CLOUD  |  EDGE  |  IOT DEVICE in PHYSICAL ENV

Service in the Cloud

Deployed Application

IoT Devices and Physical Environment

Direct conflict

Indirect conflict
Design of reliable Actuation Conflict Managers in a specific case and DEVS

- Safe Actuation Conflict Manager Assisted Design

- Temporal constrains
- Logical constrains
- ACM description with ECA Rules

- ACM FSM Model
- Model Checking

- ACM Devs Model
- Test and Simulation

- ACM Synthesis
- ACM Component

Safety and Trustworthiness
Semi-automated test suite design using inverse modeling methods

Requirements for new Protocol message set

Design using UML/SysML State charts, Class diagrams and Interaction diagrams

Translation into DEVS and SES equivalents

Simulation based testing in net-centric environment

standards conformance testing

Distributed Interactive Simulation (DIS)/Standard Interface for Multiple Platform Link Evaluation (SIMPLE)

Remote testing of implementations over the Web

MS4 Me DEVS IDE

Rational Rhapsody Development Environment

MBSE and DEVS: Development and Test of New Message Set for Tactical Data Link Standard

Semi-automated test suite design using inverse modeling methods

Translation into DEVS and SES equivalents

Simulation based testing in net-centric environment

standards conformance testing

Distributed Interactive Simulation (DIS)/Standard Interface for Multiple Platform Link Evaluation (SIMPLE)

Remote testing of implementations over the Web

MS4 Me DEVS IDE

Rational Rhapsody Development Environment

Requirements for new Protocol message set

Design using UML/SysML State charts, Class diagrams and Interaction diagrams
Automated Testing of Message Protocol Standards

**Approach:**
Automate Testing

- Translate from UML to DEVS
- Express participant systems as DEVS (dynamic, stochastic) models
- Create test models using inverse construction
- DEVS simulator executes models to induce PASS/FAIL behavior in SUT
- Interact with SUT over Middleware

**Goal:**
Increase the productivity and effectiveness of conformance for multi-participant scenarios

Formalized approach for converting standards documents into test models to run directly against a system, automating the process to the extent possible.
Message Protocol Standard Specification

Messages can be:
- Queries
- Commands
- Responses
- Information

<table>
<thead>
<tr>
<th>IO Pair Name</th>
<th>Message Name</th>
<th>State Name</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prepare</td>
<td></td>
<td>Prepare</td>
</tr>
<tr>
<td>2</td>
<td>DoTask</td>
<td>TransmitTask</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ReceivedTask</td>
<td>Acknowledge</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PerformTask</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

System

Input message

Output message

state
Overall Architecture of UML to DEVS Model Testing Methodology

1. **UML Model**
   - Add external Message passing functions

2. **Xmi file(s)**

3. **UML Model Parser**
   - Protocol Specification
   - Variables Parameters
   - Test Data Extraction

4. **Abstract DEVS Model for Testing**
   - DNL file(s)
   - Generation

5. **Test Driving DEVS Model**
   - Add Test Data set to the model
   - Use a simulator for socket capability

6. **Test Driving DEVS Socket Simulation Executable (Jar)**

7. **JSON message interactions**

8. **MS4Me**
   - Testing Results
   - Testing Result Display (Pass/Fail)
DEVS-Based Test Development

1. UML
   - Ready
   - Ready
   - X51
   - X52
   - X53

2. DEVS Model
   - Mapping State/Message Pairs

3. Inverse DEVS Model

4. Test Driver
## Mapping State/Message pairs to DEVS

### UML to DEVS Model

<table>
<thead>
<tr>
<th>Transaction</th>
<th>state</th>
<th>message</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation</td>
<td>X</td>
<td></td>
<td>“waitFor” + X</td>
<td>“send”+X(transaction number) +”_Result”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“?”+ X</td>
<td>“!”+ X(transaction number) +”_Result”</td>
</tr>
<tr>
<td>Transmission</td>
<td>X</td>
<td>M</td>
<td>waitFor” + X</td>
<td>“send”+M</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“?”+ X</td>
<td>“!”+M+”_Msg”,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“send”+ X(transaction number) +”_Result”</td>
<td>“!”+ X(transaction number) +”_Result”</td>
</tr>
<tr>
<td>Reception 1</td>
<td>X</td>
<td>M</td>
<td>“waitFor” + M</td>
<td>“send”+ X(transaction number) +”_Result”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“?”+ M+”_Msg”</td>
<td>“!”+ X(transaction number) +”_Result”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“?”+ X</td>
<td>“!”+ X(transaction number) +”_Result”</td>
</tr>
<tr>
<td>Reception 2</td>
<td>X</td>
<td></td>
<td>waitFor” + X</td>
<td>send+X (transaction number) +”_Result”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“?”+ X</td>
<td>“!”+ X(transaction number) +”_Result”</td>
</tr>
</tbody>
</table>
DEVS models Derived From UML Specification

Example State/Message Pairs

<table>
<thead>
<tr>
<th>Pair Name</th>
<th>State Name</th>
<th>Message Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TransmitTask</td>
<td>DoTask</td>
</tr>
<tr>
<td>3</td>
<td>Acknowledge</td>
<td>ReceivedTask</td>
</tr>
<tr>
<td>4</td>
<td>PerformTask</td>
<td></td>
</tr>
</tbody>
</table>
Inverse DEVS Model becomes Test Driver

<table>
<thead>
<tr>
<th>Conversion</th>
<th>Abstract DEVS model (from)</th>
<th>Inverse DEVS Test Model (to)</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>state</td>
<td>TESTstate</td>
</tr>
<tr>
<td>TA</td>
<td>Infinity</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Not Infinity</td>
<td>Infinity</td>
</tr>
<tr>
<td>Message</td>
<td>Output Message</td>
<td>Input Message</td>
</tr>
<tr>
<td></td>
<td>Input Message</td>
<td>Output Message</td>
</tr>
</tbody>
</table>
Run a coupled model with PES

MS4Me automatically generates toJson and setJson functions for DEVS messages.

Simulation output

[80.0, 9:35:54.28] Testing Finished!
[80.0, 9:35:54.28] Passed: 3 out of total test data: 4
[80.0, 9:35:54.28] Testing Result: Passed rate: 75.0%
Distributed Simulation Protocol

- `getNextEventTime()`
- Clock for Next Event time
- `injectInput(final double simulationTime, final MessageBag injectedInput)`
- `setLastOutput(getRootSimulator().computeOutput())`
- Create Json message from message bag
- `getRootSimulator().executeNextEvent(getCurrentSimulationTime())`

Time unit: second

Input message: From socket

Output message: To socket

Input from outside

TA = 0
Operational Test Driver Environment

<table>
<thead>
<tr>
<th>Pair Name</th>
<th>State Name</th>
<th>Message Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TransmitTask</td>
<td>DoTask</td>
</tr>
<tr>
<td>3</td>
<td>Acknowledge</td>
<td>ReceivedTask</td>
</tr>
<tr>
<td>4</td>
<td>PerformTask</td>
<td></td>
</tr>
</tbody>
</table>

System Entity Structure

Pruning & generation

Test Cases

Parameters

Values

Prepare

DoTask

DEVS Message

UML Message

Socket Communication
RTSync’s Projects illustrate DEVS Applications

- Model–Level Integrated Simulation Architecture for Collaborative Development
- Automated Selection/Construction of Scenarios for Multi-missile engagements
- Methodology for automated test of data link protocols
- Generation of labelled training data for deep learning of radar signatures
- Value-based Pathways-based Coordination of health care
Model–Level Integrated Simulation Architecture for Collaborative Development

CAMLI Graphical User Interface (GUI)

- Model browsing
- Model reuse
- Model editing
- Model composition

Network (Cloud)

Model repository
- Algorithmic models
- Atomic models
- Coupled models
- Interface definitions

Simulation repository
- Simulation results
- Unit tests
- Verification History

DEVS Framework Services
- SES
- DEVS simulator

MS4 Me (current status)
Model continuity: Transform models for real time application

- Same DEVS models can be designed, evaluated and transformed to operating system for actual real application.

**Phase 1: Simulation by simulation clock**
- Simulate models and SES with simulation clock
- Evaluate various design specifications and performance

**Phase 2: Simulation by real-time clock**
- Continue to use same models of phase 1
- Run simulation with real-time clock
- Evaluate real-time characteristics of system

**Phase 3: Transform models to real application**
- Continue to use same models of phase 2
- Directly interface to real world application
Books:

“Model Engineering for Simulation”, Editors: L. Zhang, B. P. Zeigler, and L. Lian


“Modeling and Simulation-based Data Engineering Introducing Pragmatics into Ontologies for Net-Centric Information Exchange”, B. P. Zeigler and P. Hammonds, 2007

Contacts
For more information, contact info@rtsync.com or POCs;

Dr. Bernard P. Zeigler Chief Scientist zeigler@rtsync.com
Dr. Doohwan Kim President & CEO dhkim@rtsync.com