IEEE Computer Society FREE Webinars

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

April 9, 2020

Bernard P. Zeigler
Prof. Emeritus University of Arizona
Chief Scientist, RTSync Corp.
IEEE 2019-2021 Distinguished Speaker
zeigler@ece.arizona.edu
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.
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
- Software Engineering
  - Architectural Design
  - Smart cities, AI-based technology
  - Cyber-physical systems
  - Internet of Things
- Modeling and Simulation
  - Test and evaluation, Conflict management, V&V

- LifeCycleML
- SysML
- UML
- SoSADL
- DEVS
  - Discrete Event System Specification
How Model-Based Systems Engineering (MBSE) and Modeling & Simulation (M&S) need to expand to link up:

Model-Based Systems Engineering (MBSE) should*

- Encourage development of architecture products that *directly* support development of modeling/analysis products
- Support *continuous interaction* of architecture developers and modeling and simulation developers to clearly link products with the defined problem as the focus
- Enable *system design solutions to be explicitly linked* to the defined problem as the result connected, interdependent processes

Modeling and Simulation should

- Enhance the architectural products from MBSE with *increased realism* as necessary to solve the defined problem
- Support *continuous interaction* of architecture developers and M&S developers to clearly link products with the defined problem as the focus
- Enable *continuity of elaboration and trace back* of high resolution simulation models to originating MBSE formulations

History of MBSE and Discrete Event System Specification (DEVS)

A. Wayne Wymore

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

Model-Based Systems Engineering, 1993

Systems Modeling Language (SysML), 2001

B.P. Zeigler
Theory of Modeling and Simulation, 1976 (DEVS)

Multi-formalism, Multi-Resolution Modeling

?
Types of System Specification and DEVS Computational Basis

DEVS simulates any system

DEVS specifies a Discrete Event System

Basic way to specify a system in algorithmic form

Specific types of system specification
### 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>
Morphisms and Equivalences at each Level of System Specification
Why the Discrete Event System Specification (DEVS) Formalism?

- **Math System Theory Foundation – supporting application**
  - Computational basis for System theory-based simulation
  - Closure under coupling, universality, uniqueness, relation to other formalisms
  - Hierarchical Model Construction supports SoS development
  - DEVS simulation protocol assures correctness of model execution/time management
  - Supports Hybrid-(Cyber-physical) System modeling

- **In Application to Systems of Systems (SoS)**
  - Models, Simulators and Experimental Frames are distinct entities
  - Precise and well-defined mathematical representation
  - Fosters reusable model/experiment components in repositories
  - Event-based execution offers efficiency advantages
  - DEVS software deployed on multiple single/distributed computing platforms
Multi-Resolution Model Family Development

Insert as component to create hierarchical model

System Entity Structure
System Entity Structure (SES)
Example: Healthcare System Simulation Ontology

Legend:
- Entity (something that have existence in a certain domain)
- Aspect (a way of decomposing an entity into more detailed parts)
- Specialization (alternative choices that an entity can take on)
- Multi-aspects (components are all of the one kind)

System level
- Major specializations

Facet level
- System components

Scale level
- Spatial & temporal scales

Model level
- Well-established & newly developed reusable artefacts

SES Pruned for Pathways-based care coordination of virus management

DEVS Agent Pathways Model

Viral Spread Sys. Dyn. Model
Intervention Prediction Model: DEVS/SES Healthcare Methodology

Data available for learner to tune the model to provide the best predictions of intervention outcomes.

Learns the value of interventions and the parameters/model that best predict outcomes.

Current best setting of model parameters/model structure.

Model of how the environment responds to interventions as measured by desired value.

Data available for learner to tune the model to provide the best predictions of intervention outcomes.

Learns the value of interventions and the parameters/model that best predict outcomes.

Current best setting of model parameters/model structure.

Model of how the environment responds to interventions as measured by desired value.

Real System (agents responding to intervention).

Observation measurement of value metrics (performance, cost).

Selected Intervention.
Implementation Of Intervention Model on Digital Health Infrastructure
### Continuous Improvement in Continuity of Care (System of Systems Problem)

#### Generic Pipeline Model

#### Taiwan COVID-19 Instantiation

<table>
<thead>
<tr>
<th>Characteristic feature</th>
<th>Public Health Disease Continuity of Care</th>
</tr>
</thead>
<tbody>
<tr>
<td>System of Systems</td>
<td>Stages of Continuity of Care</td>
</tr>
<tr>
<td>Collaboration requirement</td>
<td>Stages form pipeline, each stage must follow the prior one and set up the next one, hand-offs from one to the next are problematic</td>
</tr>
<tr>
<td>Component Systems</td>
<td>Stages are distinct from each other having different goals and (conceptually) well-defined interfaces</td>
</tr>
<tr>
<td>Component Specialization</td>
<td>Alternative processes for stages are specialized to support the different goals of the respective stages</td>
</tr>
<tr>
<td>Combinatorial Search</td>
<td>Continuous improvement seeks to adjust sub-processes and/or internal couplings (hand-offs) for “optimal” effectiveness/efficiency</td>
</tr>
</tbody>
</table>
1. Designer draws UML Activity Diagram
2. Diagram autogenerated DEVS
3. DEVS enhanced to Stochastic form
4. Designer sets parameter values and conducts simulation
Integration of Activity Specification into DEVS M&S Development Environment

- Digitizes documentation with well-defined semantics
- Provides well-defined primitives for ambiguous UML ones
- Improves expressive ability of primitives
- Off-loads details to down-stream DEVS
Integration of Activity Specification into DEVS M&S Software Environment (MS4 Me) - Video
Simulation-based Design of Healthcare System of Systems

Participatory Techniques to Involve Stakeholders in Modeling Activities

Multi-perspective Modeling Framework

UML Activity Diagram Transformation to DEVS Pathways
Message Protocol Development and Testing

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

<table>
<thead>
<tr>
<th>Message Name</th>
<th>State Name</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prepare</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>DoTask</td>
<td>TransmitTask</td>
</tr>
<tr>
<td>3</td>
<td>ReceivedTask</td>
<td>Acknowledge</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>PerformTask</td>
</tr>
</tbody>
</table>
UML-to-DEVS-Based Test Development

1. Design New Message Set for Jet Fighter-UAV Interaction Use Case Example (Retarget Command) In UML with IBM Rhapsody
2. Mapping State/Message Pairs (UML-to-DEV) with MS4 Me
3. Generate Test-ready model to drive test case sequences by MS4 Me
4. Test Driver model interacts with UML Executable in Rhapsody to complete test cases

Message flow between Test Driver and Rhapsody

DEVS-based Test Environment
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>Time advance</td>
<td>Infinity</td>
<td>Finite</td>
</tr>
<tr>
<td></td>
<td>Finite</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>
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

- Prepare
- DoTask

Parameters

- P1
- Pn

Values

- Values

Pruning & generation

Test Driving

- DEVS Model Set

UML Model Set

DEVS Simulation Protocol

Socket Communication
Toward Convergence of MBSE and M&S

Emerging MBSE Standards, Tools and Methods

Matured Integrated System Methods & Metrics

Architecture Models Integrated with Simulation, Analysis and Visualization

Defined MBSE Theory, Ontology and Formalisms

Distributed and Secure Model Repositories Allow Pattern Discovery for Reuse

DEVS M&S Standards

DEVS Framework Supports Integration and Coordination needed for SoS Methods

DEVS Diagrams (Activity,..) for Architecture Models

Multiformalism, multiresolution Systems representation through DEVS Experimental Frame

DEVS/SES Support of hierarchical coupled Models, reusability, and composability

International Council on Systems Engineering (INCOSE) Roadmap

Discrete Event System Specification (DEVS)
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

https://www.researchgate.net/publication/328977446_MBSE_without_Simulation_State_of_the_Art_and_Way_Foward
Backup
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

Remote testing of implementations over the Web

Rational Rhapsody Development Environment

MS4 Me DEVS IDE

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

**System Under Test (SUT)**
- Test Model
- DEVS Simulator
- Middleware
- Socket (example)

**Network**
Overall Architecture of UML to DEVS Model Testing Methodology

UML Model → UML Model Parser → Test Data Extraction

- Add external Message passing functions
- Link to Socket Communication Capability

UML Model For Testing

Protocol Specification

Variables Parameters

Decisions

Abstract DEVS Model for Testing

DNL file(s)

Creation:

Data format (JSON)

Add Test Data set to the model

Use a simulator for socket capability

Test Data Set:

Add Test Data set to the model

Socket Simulation Executable (Jar)

Testing Results:

Testing Result Display (Pass/Fail)

MS4Me
## Mapping State/Message pairs to DEVS

<table>
<thead>
<tr>
<th>Transaction</th>
<th>state</th>
<th>message</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>State</td>
<td>Input message</td>
</tr>
<tr>
<td><strong>Preparation</strong></td>
<td>X</td>
<td></td>
<td>“waitFor” + X</td>
<td>“?” + X</td>
</tr>
<tr>
<td><strong>Transmission</strong></td>
<td>X</td>
<td>M</td>
<td>“waitFor” + X</td>
<td>“?” + X</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><strong>Reception 1</strong></td>
<td>X</td>
<td>M</td>
<td>“waitFor” + M</td>
<td>“?” + M +”_Msg”</td>
</tr>
<tr>
<td><strong>Reception 2</strong></td>
<td>X</td>
<td></td>
<td>“waitFor” + X</td>
<td>“?” + X</td>
</tr>
</tbody>
</table>

### UML to DEVS Model

- State: X
- Message: M
- Input: “waitFor” + X
- Output: “send” + X(transaction number) +”_Result”
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>

Diagram showing state transition and message flow between states such as Prepare, TransmitTask, DoTask, Acknowledge, and PerformTask.
Composed DEVS Model
Implementing DEVS Test Driver Model: Creating variables and adding java library
Implementing Test Driving DEVS Model

: creating message type and code tagging
Running Test Driving DEVS Model

MS4Me automatically generates toJson and setJson functions for DEVS messages.

Run a coupled model with PES

DoTask.java
- public void setJson(String jsonMsg)
- public String toJson()

Simulation output
Distributed Simulation Protocol

Clock for Next Event time

- `getNextEventTime()`
- `getRootSimulator().executeNextEvent(getCurrentSimulationTime())`

Input from outside

- `injectInput(final double simulationTime, final MessageBag injectedInput)`
- `setLastOutput(getRootSimulator().computeOutput())`

Output message To socket

- Create Json message from message bag

- `getRootSimulator().executeNextEvent(getCurrentSimulationTime())`

Time unit: second

Input message From socket

- `injectedInput`
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