System of Systems Model
Building and Acausal Simulation Environment

Peter Menegay – Director, Funded R&D, SynaptiCAD
pete@syncad.com, 540-953-3390
Dale Burnham – Lead Systems Engineer, AFRL
dale.burnham@us.af.mil

National Defense Industrial Association (NDIA)
21st Annual Systems Engineering Conference
Oct. 25, 2018
Technology Motivation

- DoD systems are increasingly complex and challenge human cognitive and organizational abilities. We are now combining those systems into System of Systems.
  - Engineering model flexibility & robustness.
  - Reuse of models.
  - SoS requirements expression and flowdown.
  - Discovering unforeseen behavior through trade studies.
  - Understanding complex results.
  - Accurate simulations, well before we commit.
  - Handling highly scaled simulation problems – “digital twin”.
The Basics

Simulating the SoS

Creating SoS from Systems

Creating Systems from Subsystems

Models from 1st Principles

System
MQ-9 Aircraft

GBU

SAR

FLIR

System
Ground Control

Mission

System
Satellite

Communications
System Computation Platform

Infrastructure for building complex systems for simulation

Our approach is to fix weaknesses in the 3 key areas
3 Core Elements & Workflow

1. **Eqn. Based Model Creator** -- Simultaneous equation solver for creating new engineering models for subsystems.

2. **System Builder** -- User interface for creating new system models.

3. **System Simulator** -- SystemVerilog simulation engine for system model execution and results.
New Capability
Example Problem

An MQ-9 Reaper flies to a target zone and collects sensor data. It identifies a missile site, destroys it and returns to base. The aircraft is equipped with SAR, FLIR, and GBU as part of its system. The ground control station (GCS) controls mission parameters and determines probability of kill. A satellite handles communications between the two.
• First build independent systems: aircraft, ground control station, and satellite. Aircraft is composed of multiple subsystems (e.g. SAR) which can be modeled in equation solver and connected to overall system:
Next, SoS is built by specifying which systems to unite. A flat hierarchy of all contained subsystems results. Variable links must be created.
Simulation Results

- The SoS is simulated using the System Verilog engine and results are shown in a timing diagram:
Requirements Modeling

- A template is used to create requirements sentences in English which are translated to Python for analysis. Requirements can also be written directly in Python.
SoS Trade Studies

- Operate over the entire SoS and perform full simulations for each point. DOE, Monte-Carlo, and single variable optimization is available.
AADL Integration

• Simulation models are architecturally analogous to AADL language.
Acausal Solver/Driver

- Acausal simulation means that what we consider an input or output is not fixed. All models are solved as a system of equations using two methods:

**Acausal Method 1**

<table>
<thead>
<tr>
<th>EquationSolver</th>
<th>EquationSolver</th>
<th>EquationSolver</th>
</tr>
</thead>
<tbody>
<tr>
<td>unknowns₁ = subsystem₁(knowns₁, 0)</td>
<td>unknowns₂ = subsystem₂(knowns₂, 1)</td>
<td>unknowns₃ = subsystem₃(knowns₃, 0)</td>
</tr>
<tr>
<td>unknowns₂ = subsystem₂(knowns₂, 0)</td>
<td>unknowns₂ = subsystem₂(knowns₂, 1)</td>
<td>unknowns₃ = subsystem₃(knowns₃, 1)</td>
</tr>
<tr>
<td>unknowns₃ = subsystem₃(knowns₃, 0)</td>
<td>unknowns₂ = subsystem₂(knowns₂, 2)</td>
<td>unknowns₃ = subsystem₃(knowns₃, 2)</td>
</tr>
</tbody>
</table>

* etc *

**Acausal Method 2**

Equation Solver

unknowns = simulator_system_of_systems(knowns)

Simulator (System of Systems)

$ t = 0 \, \text{sec} $  $ t = 1 \, \text{sec} $  $ t = 2 \, \text{sec} $
Acausal Method 1

- Each system is rendered as an equation to be solved at every timestep. The simulator is the driver and the equation solver runs behind the scenes.
Acausal Method 2

- The entire SoS is viewed as an equation to be solved over the entire span of time of the simulation. The equation solver is the driver and the simulator runs behind the scenes.
Some Observations

• Scale up.
  – Human scalability: our ability to see, understand, and manipulate large SoS models.
  – Computational scalability: Multiple nested engineering models, solved iteratively by equation solver, are possible. Trade studies make this worse.

• “Explosion” of data caused by trade studies.

• English requirements are backed up by requirements expression through a computer language (i.e. Python).

• Requirements modeling can be enforced through equation solver.

• The acausal capability allows for many scenarios without having to rebuild the model. Inputs/outputs can be switched.

• This tool is designed to be linked from 3rd party systems engineering tools such as AADL or SysML.
Backup Slides
1. Eqn. Based Model Creator

A general purpose model-creation environment for engineering analysis. Under development.

- Solves any system of nonlinear simultaneous equations.
- Manages the core numerical library to achieve robustness.
- Generates Python functions to use in simulation code.
- Uses a library concept for storing functions for later use.
2. System Builder

A GUI that helps you build, connect, and modify sub-system models for simulation.

- Create system models from a library of pre-built subsystems.
- Create subsystem models from equation solver and external tools.
- Publish subsystems to library.
- Easily replace subsystems with higher/lower fidelity ones.
2. System Builder, cont.

Includes a feature to help you edit calculations or link to them from external tools.

Link to pre-built models in SolverCAD, ModelCenter, Excel, or your own program.

Buttons for quickly creating common Python constructs.

Edit the calculation file for any subsystem in Python.
3. System Simulator

The system simulator combines a high-performance compiled-code SystemVerilog simulator with a Python interpreter to enable engineering level modeling of real world systems.

- Timing diagram with simulation results
- Generated SystemVerilog code.
- Hierarchical view of subsystems and components.
- Full IDE including single step debugging, breakpoints, etc.
- Design browsing & navigation.
- Various output formats
Multifidelity Modeling

- Models of different fidelity can be switched on the fly.
- As the project advances, the simulation environment remains in place, and maintains connectivity with previous models.

Replacement rules for switching model fidelity

Variable mapping to ensure continuity between models
Model Libraries

• Subsystem model library.
  – Orbit calculations, solar panel, battery, etc. are publishable and retrievable from library.

• Function library for equation solver.
  – Generated functions can be accessed by System Builder.
Synthesis

- Compares simulation subsystems with a catalog of parts.
  - User has presumably optimized the subsystem and now wants to select hardware.
  - Software will choose the closest part from catalog and resimulate.
Satellite Model

Satellite circles the earth in a standard elliptical orbit. It’s mission is to collect earth data over an experimental zone and download it to a ground station at another location. It charges a battery in the sun and depletes the battery in the shade. The simulation objective is to understand if the subsystems are sized properly.
Results

Battery slowly drains to 0
Solar Panel does not recharge it when exposed to sun
I.e., the Solar Panel is undersized. Battery is oversized.
Results, cont.

- One way to vary the solar panel / battery size is to use constrained randomization.
- Solution was to increase the solar panel area from 3.0 to 4.0 m**2 and decrease the battery capacity from 360,000 to 60,000 amp-sec.
Results, cont.

- This could also have been achieved by driving the simulator from a ModelCenter DOE.
Overall Results

- Once engineering models were made, system integration was fast, 1-2 days for this case.
  - Model libraries were key.

- Provision for multi-fidelity model switching allowed project to remain within a single environment throughout its life.

- Scalability tests on a simple vehicle object lends credence to the SystemVerilog approach.
  - SystemVerilog can simulate up to memory limits of computer. 18 million vehicles for 32-bit and 40 million for 64 bit.
  - SimPy by contrast could simulate 900,000 such objects.

- Runs could be made faster by using event-driven simulation. A 10 fold speed up was achieved this way.
  - Important for long run times over the life of the system.

We thank the NASA SBIR program for sponsoring this work.
Contract NNX15CP26P.