75 Fifth Street NW, Suite 312
Atlanta, GA 30308, USA
voice: +1-404-592-6897
web: www.Intercax.com
email: info@Intercax.com

Rose Yntema, Intercax LLC

Technote: Syndeia Simulink Capabilities

Part 2 – SysML → Simulink Executable Model

Introduction

A crucial thread in enabling model-based systems engineering (MBSE) for next-generation, complex systems is to analyze system architecture by means of simulations and verify requirements continuously during design and development phases. The general steps in this iterative simulation-based design approach are as follows:

(1) Define system architecture (design model)
(2) Create a simulation model
(3) Run the simulation model
(4) Verify requirements using simulation results
(5) Refine system architecture and repeat

However, this process often becomes challenging because: (1) system architecture is often defined by system engineers and designers who are not simulation experts and may not have the necessary skills to define and execute simulation models, (2) the communication between system architects and simulation experts is often document-based and hence laborious and error-prone.

Syndeia, our platform for MBE/MBSE, addresses these challenges by providing capabilities to (1) generate simulation models from design models using model transformations, (2) maintain a connection between elements in the design model and simulation model, (3) provide services to compare and synchronize design and simulation models bi-directionally as they evolve concurrently, and (4) reverse engineer design models from simulation models for organizations transitioning to MBSE. This installment continues our series of technical notes where we exemplify the detailed use cases in this approach, with SysML for representing system design and Simulink for representing simulation models

In this series we highlight various capabilities of Syndeia for using Simulink with SysML to coordinate the simulation-based design process of a system. Over these three initial installments, we have begun to outline scenarios for using Syndeia 2.0 to generate, connect, and compare Simulink and SysML models. Part 1 showed how SysML block and activity structures can be used to generate Simulink model reference structures, including both atomic and multi-signal ports. Part 2 will describe how to generate Simulink models from SysML with specific blocks in the Simulink library that are then executed using a MATLAB script. In Part 3 the reverse will be demonstrated; using Simulink model and block structures to generate SysML block and activity structures. In all three use cases, we also use the connections created during the generation process to compare and identify changes made on either the SysML side or Simulink side.

In this second installment of the series, we will explore the use case where an expert in both SysML and Simulink can create models in SysML which can be directly translated to Simulink and immediately executed using a script. Meanwhile, persistent connections between SysML elements and Simulink models will have been created by Syndeia, as in the first installment, so that later on as changes are made to either side, we will compare across those connections to show what changes have been made and whether the ports and interfaces remain in sync.

Generate specific blocks from the Simulink block library

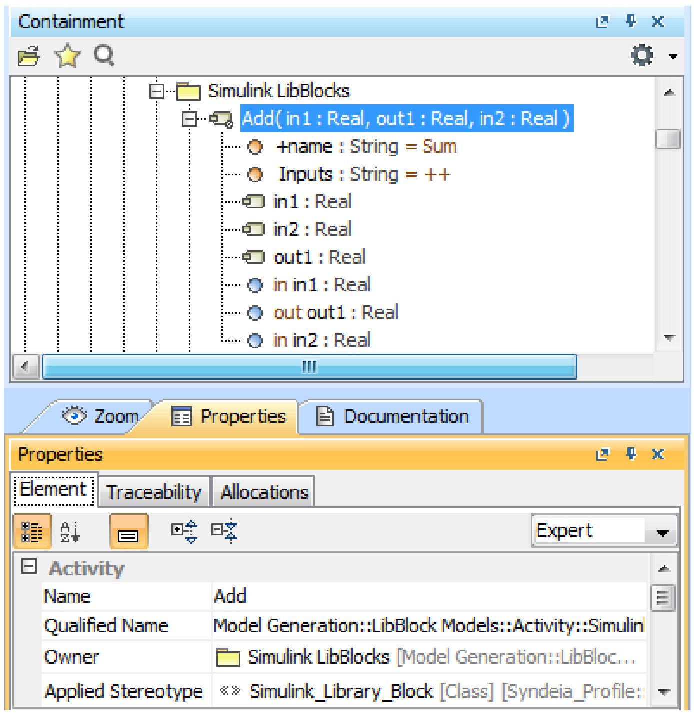
The initial goal of this scenario is to create source models in SysML, which directly translate to Simulink models and are then executed with a script. This section will detail how an expert in both SysML and Simulink can create the SysML source model and generate the corresponding Simulink model, with script-based execution of that model following in the next section.

Creating SysML elements that correspond to Simulink library blocks

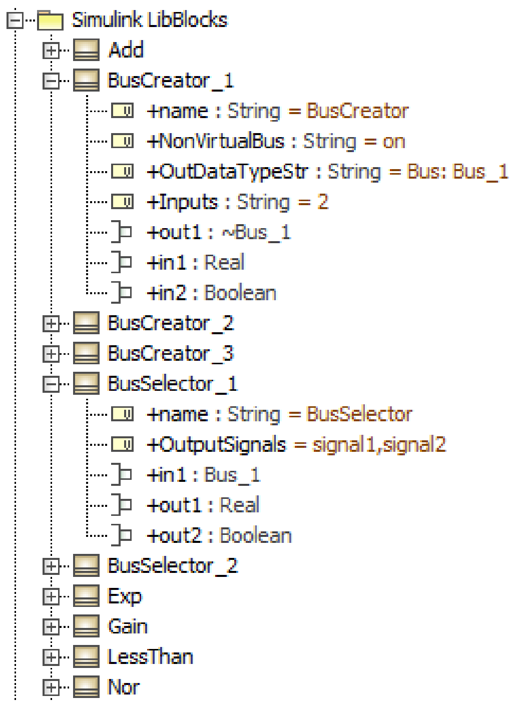
First, libraries of SysML activities (example in Figure 1) or blocks (examples in Figure 2) can be created to represent various blocks from the Simulink library, with properties configured to set their various relevant parameters[1]. We apply the **Simulink_Library_Block** stereotype from the **Syndeia** profile to each of these blocks or activities to indicate that these should *not* be created as model reference blocks (as shown in the first installment of this series), but that a specific block from the Simulink block library should be generated with any specified parameter settings. This stereotype can be seen in the **Applied Stereotype** property at the bottom of Figure 1. The correct number of input and output ports/nodes should be created in SysML corresponding to the number of inports and outports of the block in Simulink, so that the block can be properly connected to other blocks. All Simulink defaults will apply to these blocks, so activities or blocks with any parameters that need to be set differently should

[1] Block-specific parameters: http://www.mathworks.com/help/simulink/slref/block-specific-parameters.html
Parameters common to all blocks: http://www.mathworks.com/help/simulink/slref/common-block-parameters.html

have properties of the activity, or value properties of the block, of type **String**, with the default value of the SysML property equal to the value needed to programmatically set that parameter, i.e. for the **Add** activity below, the parameter **'Inputs'** was created with default value **'++'** to indicate that there are two inputs that should be added together. If parameters are set that affect the number of inports and/or outports, we must make sure that this matches the number of input and output ports/nodes. Since each instance of a Simulink block can be customized and changed like this, we may have multiple activities/blocks that correspond to the same basic Simulink block. Because of this, the name of the activity/block will *not* be used for the Simulink model. Instead a property called **"name"** of type **String** should be added for every element with the **Simulink_Library_Block** stereotype applied, and the default value of that property should be *exactly* the name of the Simulink library block type (as shown in parentheses in the list of block-specific parameters[1]), i.e. **name = 'Sum'** for the **Add** activity or **name = 'BusCreator'** for the **BusCreator_1** block.


```
Containment
Simulink LibBlocks
  Add( in1 : Real, out1 : Real, in2 : Real )
    +name : String = Sum
    Inputs : String = ++
    in1 : Real
    in2 : Real
    out1 : Real
    in in1 : Real
    out out1 : Real
    in in2 : Real

Zoom | Properties | Documentation
Properties
Element | Traceability | Allocations          Expert
Activity
Name                Add
Qualified Name      Model Generation::LibBlock Models::Activity::Simulin
Owner               Simulink LibBlocks [Model Generation::LibBloc...
Applied Stereotype  Simulink_Library_Block [Class] [Syndeia_Profile:
```
Figure 1 SysML activity for generating *Sum* library block in Simulink


```
Simulink LibBlocks
  Add
  BusCreator_1
    +name : String = BusCreator
    +NonVirtualBus : String = on
    +OutDataTypeStr : String = Bus: Bus_1
    +Inputs : String = 2
    +out1 : ~Bus_1
    +in1 : Real
    +in2 : Boolean
  BusCreator_2
  BusCreator_3
  BusSelector_1
    +name : String = BusSelector
    +OutputSignals = signal1,signal2
    +in1 : Bus_1
    +out1 : Real
    +out2 : Boolean
  BusSelector_2
  Exp
  Gain
  LessThan
  Nor
```
Figure 2 SysML blocks for generating library blocks

Using Simulink library blocks specific to signal bus models

The **BusCreator** and **BusSelector** blocks pictured in Figure 2 above are a special case of library block, used only in the case where buses are being used (as in scenario 2 of the first installment). The **BusSelector** block extracts signals *from* a bus, and the **BusCreator** block forms multiple signals *into* a bus, often for the purpose of using individual signals with other blocks that don't take buses as inputs, such as the activity model shown below in Figure 3 , in which the first signal is selected from **Bus_1**, passed through a **Sin** block, and then recombined into **Bus_1** again. Different blocks must be created to handle different buses, as the block parameters will indicate the signals and/or bus to be created or selected from. These parameters *must* match with the bus types being used. For the **BusSelector_1** block in Figure 2, the **'OutputSignals'** parameter is set to **'signal1,signal2'**, which matches the signals in

Bus_1, the bus definition element that types **in1** port. For **the BusCreator_1** block, multiple parameters are set that define the bus properties – **'Inputs' = '2'** gives the number of signals being used to create the bus, **'Bus: Bus_1'** indicates that **Bus_1** is the name of the bus to be created, and the **'NonVirtualBus'** setting indicates how Simulink will see and use the resulting bus in simulations. Note that the input and output ports of these blocks have the correct types so that the IBD created from them can be correct and consistent, but as far as Simulink is concerned, the *parameters* are what is used to define the block.

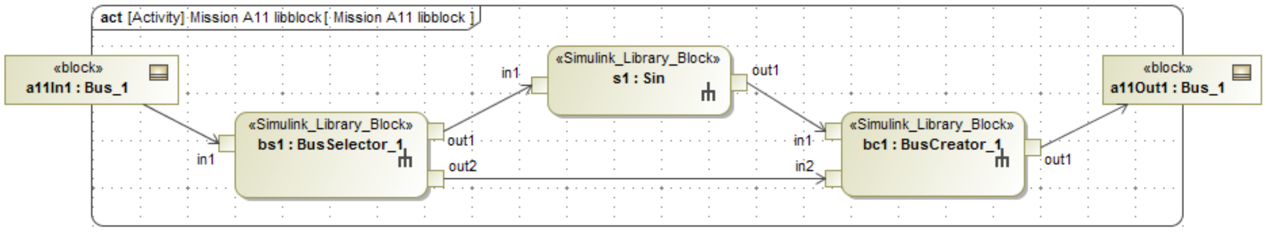
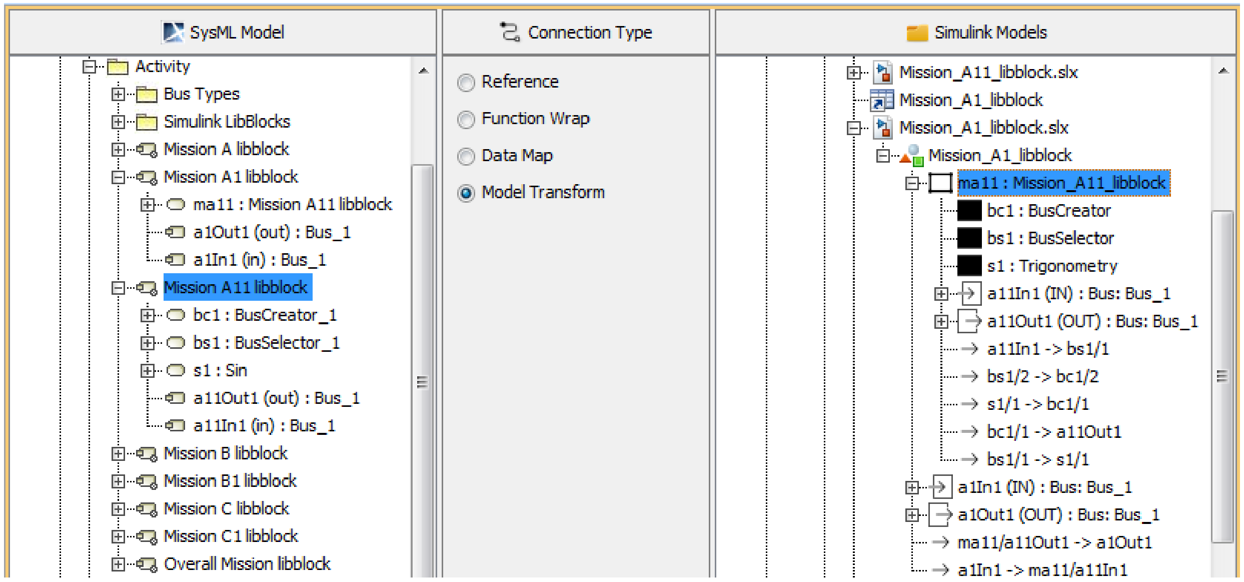


Figure 3 Activity model using the Simulink_Library_Block **stereotype for bus-related blocks and a trigonometry block**

Generating the executable Simulink model with native library blocks

As with the skeletal model in the first installment, the top-level activity/block is dragged in the Syndeia dashboard from SysML Model to a folder in a local file system repository to generate Simulink models (SysML activity structure → Simulink shown in Figure 4, SysML block structure → Simulink shown in Figure 5). Model reference blocks are shown with white boxes as before (i.e. **ma11 : Mission_A11_libblock**), but this time we can see that the Simulink library blocks described above have been generated and displayed as black boxes to signify that we can't expand the structure any further down. The reason that the **s1 : Trigonometry** block in *Simulink* corresponds to the **s1 : Sin** call behavior action in *SysML* is that the **name** property of the **Sin** activity was set to **'Trigonometry'** and then another property, **Operator = 'sin'**, set the trigonometric relationship to be used in Simulink.



SysML Model

```
Activity
  Bus Types
  Simulink LibBlocks
  Mission A libblock
  Mission A1 libblock
    ma11 : Mission A11 libblock
    a1Out1 (out) : Bus_1
    a1In1 (in) : Bus_1
  Mission A11 libblock
    bc1 : BusCreator_1
    bs1 : BusSelector_1
    s1 : Sin
    a11Out1 (out) : Bus_1
    a11In1 (in) : Bus_1
  Mission B libblock
  Mission B1 libblock
  Mission C libblock
  Mission C1 libblock
  Overall Mission libblock
```

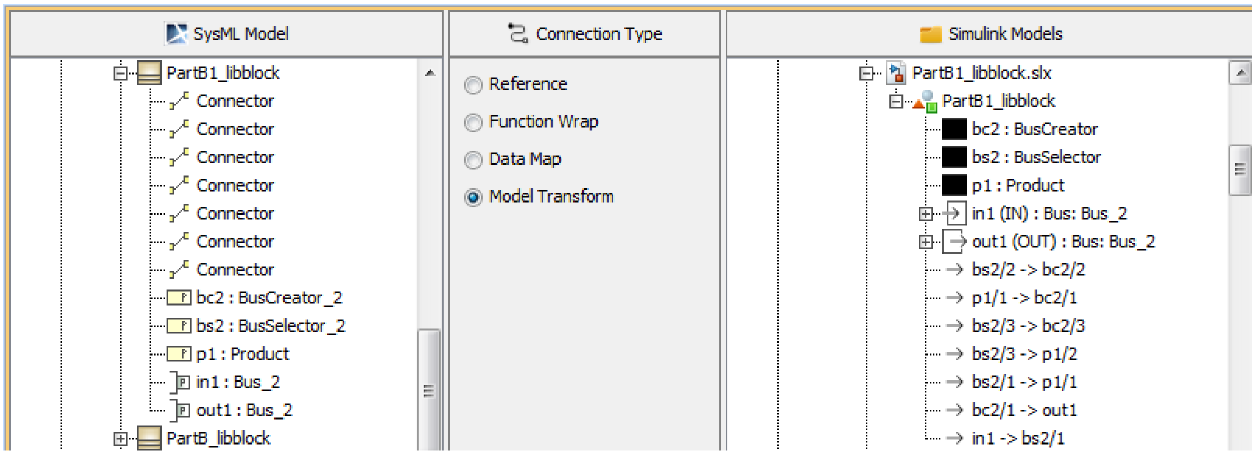
Connection Type

Reference
Function Wrap
Data Map
[✓ Model Transform]

Simulink Models

```
Mission_A11_libblock.slx
Mission_A1_libblock
Mission_A1_libblock.slx
  Mission_A1_libblock
    ma11 : Mission_A11_libblock
      bc1 : BusCreator
      bs1 : BusSelector
      s1 : Trigonometry
      a11In1 (IN) : Bus: Bus_1
      a11Out1 (OUT) : Bus: Bus_1
      a11In1 -> bs1/1
      bs1/2 -> bc1/2
      s1/1 -> bc1/1
      bc1/1 -> a11Out1
      bs1/1 -> s1/1
    a1In1 (IN) : Bus: Bus_1
    a1Out1 (OUT) : Bus: Bus_1
    ma11/a11Out1 -> a1Out1
    a1In1 -> ma11/a11In1
```

Figure 4 SysML activity structure on the left and Simulink model structure generated from SysML on the right



SysML Model	Connection Type	Simulink Models
PartB1_libblock	○ Reference	PartB1_libblock.slx
Connector	○ Function Wrap	PartB1_libblock
Connector	○ Data Map	bc2 : BusCreator
Connector	● Model Transform	bs2 : BusSelector
Connector		p1 : Product
Connector		in1 (IN) : Bus: Bus_2
Connector		out1 (OUT) : Bus: Bus_2
Connector		bs2/2 -> bc2/2
bc2 : BusCreator_2		p1/1 -> bc2/1
bs2 : BusSelector_2		bs2/3 -> bc2/3
p1 : Product		bs2/3 -> p1/2
in1 : Bus_2		bs2/1 -> p1/1
out1 : Bus_2		bc2/1 -> out1
PartB_libblock		in1 -> bs2/1

Figure 5 SysML internal block structure on the left and Simulink model structure generated from SysML on the right

As shown in the first installment, we may open the newly-generated Simulink models from Syndeia to view and rearrange them in the native MATLAB environment as shown in the figures below.

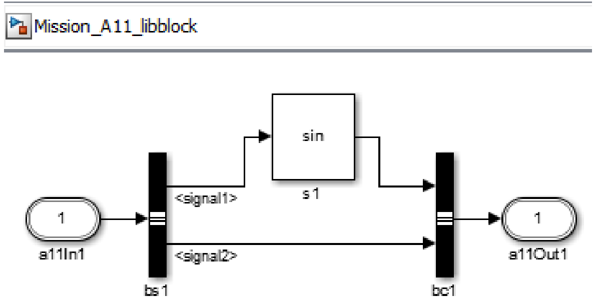


Figure 6 Mission_A11_libblock.slx (file from Figure 4)

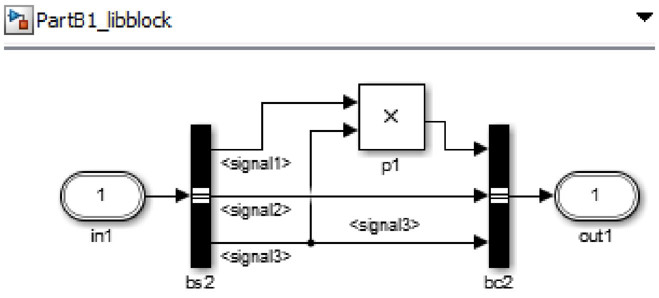
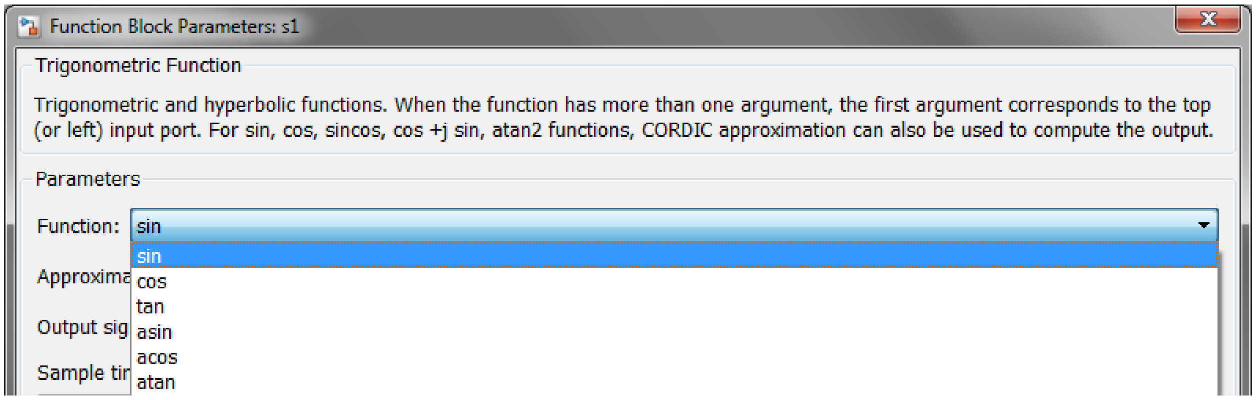


Figure 7 PartB1_libblock.slx (file from Figure 5)

To see the same parameters that were set using properties in SysML, simply double-click that block, like the **s1** block above, whose **Function** parameter is shown in Figure 8 set to **'sin'**, and where the drop-down menu here gives the other types of trigonometric functions that could be represented by the same block just by changing this parameter.



Function Block Parameters: s1

Trigonometric Function

Trigonometric and hyperbolic functions. When the function has more than one argument, the first argument corresponds to the top (or left) input port. For sin, cos, sincos, cos +j sin, atan2 functions, CORDIC approximation can also be used to compute the output.

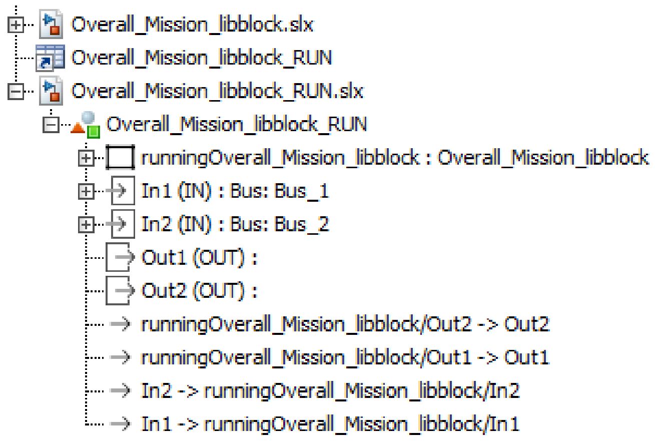
Parameters

Function: sin

Figure 8 Setting trigonometry block parameter "Function" using the Simulink user interface

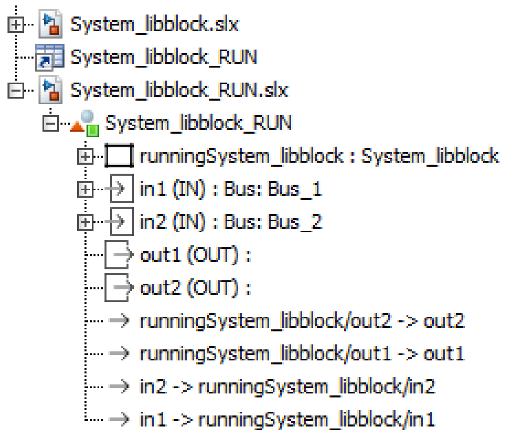
Execute Simulink model generated from SysML using a script

Next, we show how to use a MATLAB script to execute the models generated in the section above. When Syndeia generates an entire model structure, it uses inports and outports at each level, so that they can be used modularly, where a model at any given level may be used as a model reference block in a higher level model, or run as a standalone model. However, inports and outports have some limitations in MATLAB as far as how some signals may be passed into and out of the top-level model, so Syndeia has also been designed to generate a wrapper model with **"_RUN"** appended to the top-level model's name, and with **FromWorkspace** and **ToWorkspace** blocks connected to the original top-level model's inports and outports, respectively, as this model is used as a model reference block in the **"_RUN"** model , pictured in Figure 9 and Figure 10.



```
Overall_Mission_libblock.slx
Overall_Mission_libblock_RUN
Overall_Mission_libblock_RUN.slx
  Overall_Mission_libblock_RUN
    runningOverall_Mission_libblock : Overall_Mission_libblock
    In1 (IN) : Bus: Bus_1
    In2 (IN) : Bus: Bus_2
    Out1 (OUT) :
    Out2 (OUT) :
    runningOverall_Mission_libblock/Out2 -> Out2
    runningOverall_Mission_libblock/Out1 -> Out1
    In2 -> runningOverall_Mission_libblock/In2
    In1 -> runningOverall_Mission_libblock/In1
```

Figure 9 Wrapper file generated from activity model



```
System_libblock.slx
System_libblock_RUN
System_libblock_RUN.slx
  System_libblock_RUN
    runningSystem_libblock : System_libblock
    in1 (IN) : Bus: Bus_1
    in2 (IN) : Bus: Bus_2
    out1 (OUT) :
    out2 (OUT) :
    runningSystem_libblock/out2 -> out2
    runningSystem_libblock/out1 -> out1
    in2 -> runningSystem_libblock/in2
    in1 -> runningSystem_libblock/in1
```

Figure 10 Wrapper file generated from block model

Seen in Simulink, the wrapper models look like this:

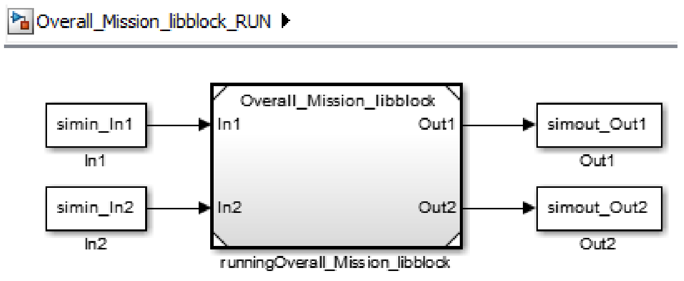


Figure 11 Overall_Mission_libblock_RUN.slx (file from Figure 9)

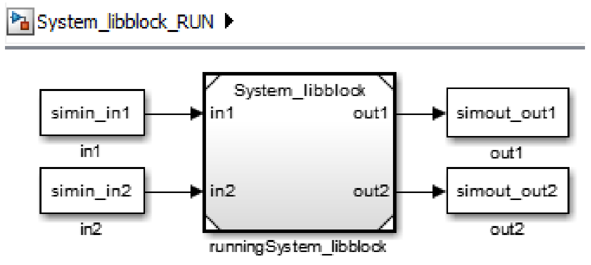
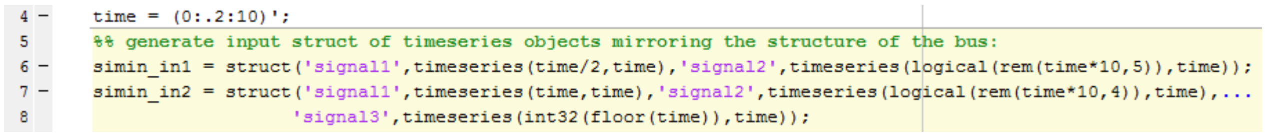


Figure 12 System_libblock_RUN.slx (file from Figure 10)

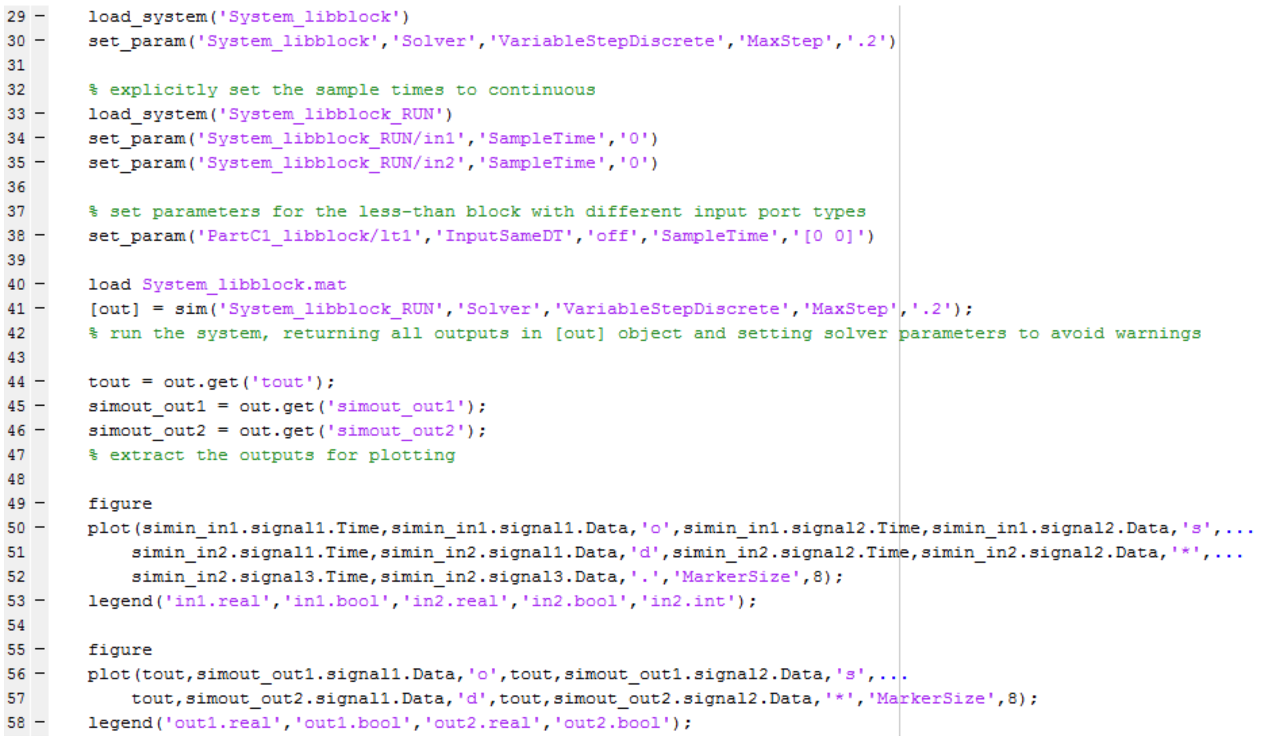
Using this wrapper model for inputs and outputs, we can write a script (i.e. **Run.m**) to set up inputs, run this set of models, and generate plots of inputs and outputs. First, we set the variables for the inputs and load them into the workspace, which is shown in Figure 13.



```
time = (0:.2:10)';
%% generate input struct of timeseries objects mirroring the structure of the bus:
simin_in1 = struct('signal1',timeseries(time/2,time),'signal2',timeseries(logical(rem(time*10,5)),time));
simin_in2 = struct('signal1',timeseries(time,time),'signal2',timeseries(logical(rem(time*10,4)),time),...
                   'signal3',timeseries(int32(floor(time)),time));
```

Figure 13 Excerpt from Run.m where input signals are set up and loaded into the MATLAB workspace

Next in the script, we set the solver and maximum step size to be used for each model that was used as a model reference block, including the top-level model shown at line 30 in Figure 14. In order to set parameters, each model must be loaded first (line 29). We also set continuous sample times for inputs to the top-level model (lines 34 and 35) and set any other parameters that need to be set; in this case we change the settings of our *less-than* block so that the two inputs can be different data types, and we also set it to have a continuous sample time (line 38). Then we load the **.mat** file that contains bus definitions (line 40) and run a simulation of the model (line 41) using the inputs that have previously been loaded into the workspace. From the collective output, we extract the various individual outputs (lines 44-46) and plot the inputs and outputs separately (lines 49-58).



```
29 -    load_system('System_libblock')
30 -    set_param('System_libblock','Solver','VariableStepDiscrete','MaxStep','.2')
31
32      % explicitly set the sample times to continuous
33 -    load_system('System_libblock_RUN')
34 -    set_param('System_libblock_RUN/in1','SampleTime','0')
35 -    set_param('System_libblock_RUN/in2','SampleTime','0')
36
37      % set parameters for the less-than block with different input port types
38 -    set_param('PartC1_libblock/lt1','InputSameDT','off','SampleTime','[0 0]')
39
40 -    load System_libblock.mat
41 -    [out] = sim('System_libblock_RUN','Solver','VariableStepDiscrete','MaxStep','.2');
42      % run the system, returning all outputs in [out] object and setting solver parameters to avoid warnings
43
44 -    tout = out.get('tout');
45 -    simout_out1 = out.get('simout_out1');
46 -    simout_out2 = out.get('simout_out2');
47      % extract the outputs for plotting
48
49 -    figure
50 -    plot(simin_in1.signal1.Time,simin_in1.signal1.Data,'o',simin_in1.signal2.Time,simin_in1.signal2.Data,'s',...
51          simin_in2.signal1.Time,simin_in2.signal1.Data,'d',simin_in2.signal2.Time,simin_in2.signal2.Data,'*',...
52          simin_in2.signal3.Time,simin_in2.signal3.Data,'.','MarkerSize',8);
53 -    legend('in1.real','in1.bool','in2.real','in2.bool','in2.int');
54
55 -    figure
56 -    plot(tout,simout_out1.signal1.Data,'o',tout,simout_out1.signal2.Data,'s',...
57          tout,simout_out2.signal1.Data,'d',tout,simout_out2.signal2.Data,'*','MarkerSize',8);
58 -    legend('out1.real','out1.bool','out2.real','out2.bool');
```

Figure 14 Excerpt from Run.m **setting model and block parameters, running the simulation, and creating relevant plots**

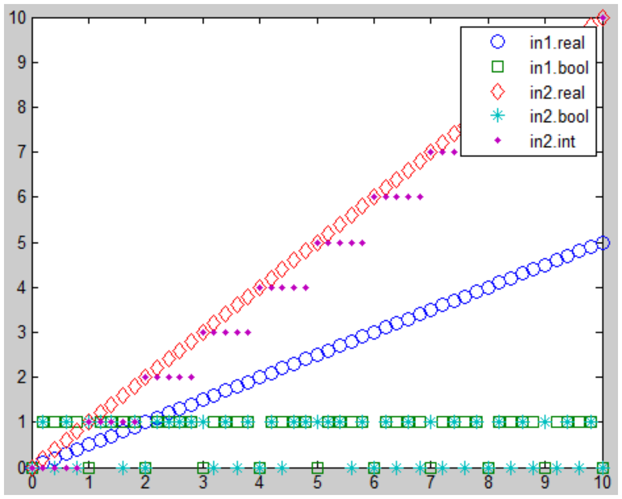


Figure 15 Input plots (lines 49-53 in Figure 14)

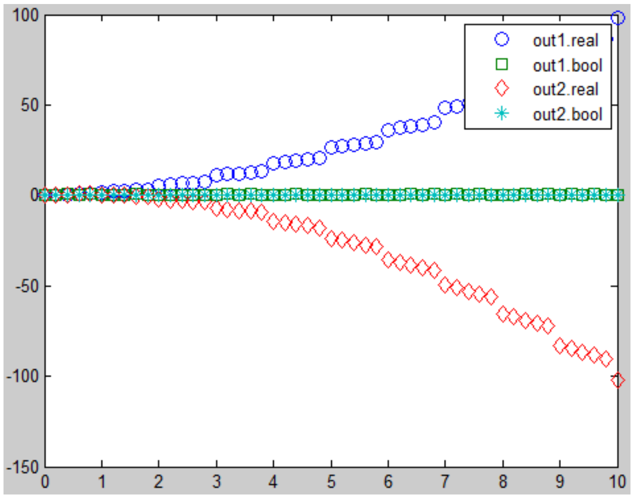
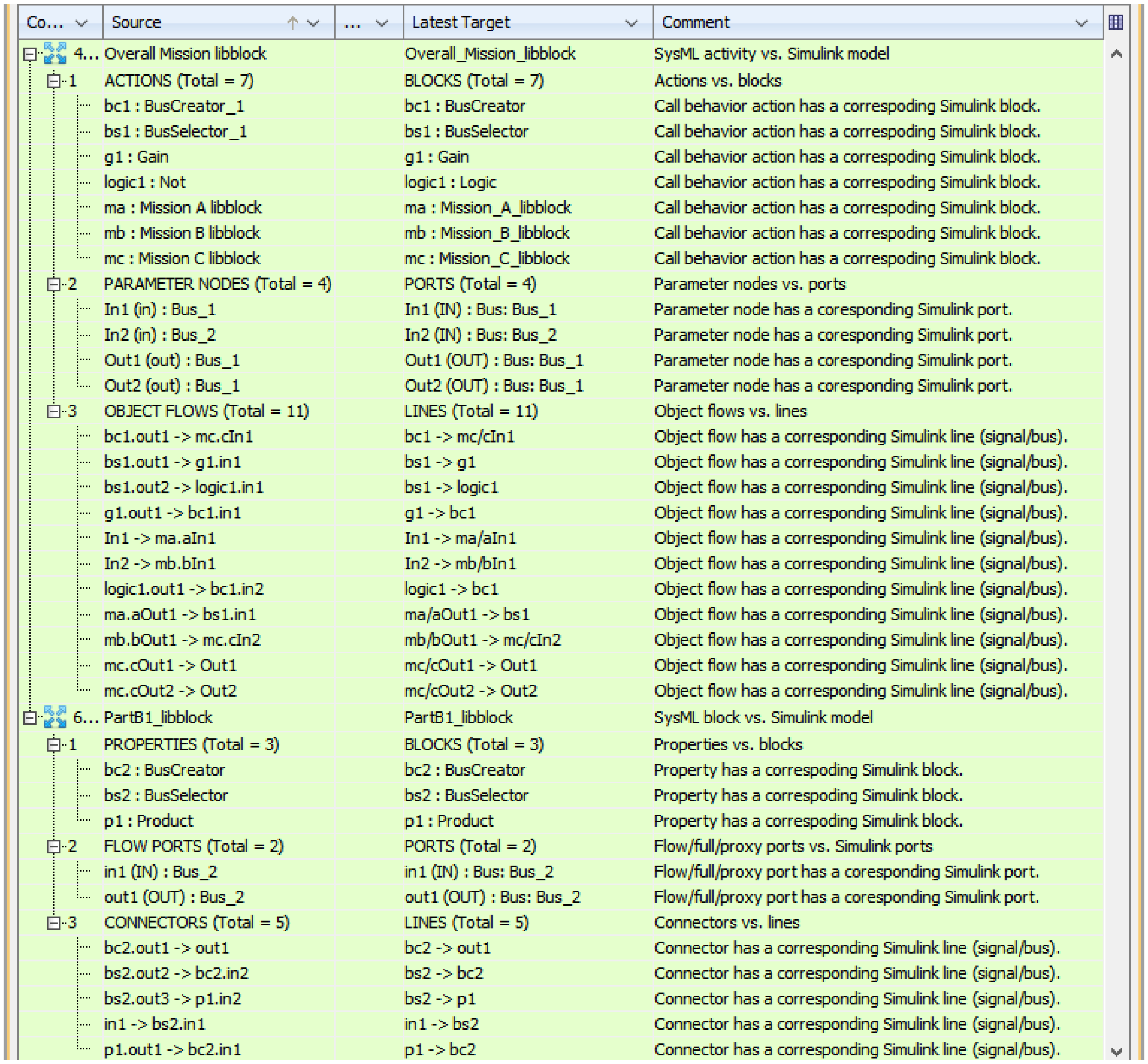


Figure 16 Output plots (lines 55-58 in Figure 14)

The plots in Figure 15 and Figure 16 show first the inputs generated in the script pictured above and next the outputs generated by running these inputs through the Simulink model.

Compare Simulink models with their SysML sources

Finally, as we did in the first installment, we will utilize the persisted connections to compare the Simulink models and SysML blocks/activities. First, we will compare some connections before changing anything to show what happens when the models are in sync. The Comparison Result shown in Figure 17 verifies in this case that all elements are in sync as they were just created. The different numbers, names, and types of each set of actions, parameter nodes, and object flows (for activities) or properties, flow ports, and connectors (for blocks) are shown below the main connection heading, and the comment column gives further details on the status of each SysML and/or Simulink element.



Co...	Source	...	Latest Target	Comment
4...	Overall Mission libblock		Overall_Mission_libblock	SysML activity vs. Simulink model
1	ACTIONS (Total = 7)		BLOCKS (Total = 7)	Actions vs. blocks
	bc1 : BusCreator_1		bc1 : BusCreator	Call behavior action has a correspoding Simulink block.
	bs1 : BusSelector_1		bs1 : BusSelector	Call behavior action has a correspoding Simulink block.
	g1 : Gain		g1 : Gain	Call behavior action has a correspoding Simulink block.
	logic1 : Not		logic1 : Logic	Call behavior action has a correspoding Simulink block.
	ma : Mission A libblock		ma : Mission_A_libblock	Call behavior action has a correspoding Simulink block.
	mb : Mission B libblock		mb : Mission_B_libblock	Call behavior action has a correspoding Simulink block.
	mc : Mission C libblock		mc : Mission_C_libblock	Call behavior action has a correspoding Simulink block.
2	PARAMETER NODES (Total = 4)		PORTS (Total = 4)	Parameter nodes vs. ports
	In1 (in) : Bus_1		In1 (IN) : Bus: Bus_1	Parameter node has a coresponding Simulink port.
	In2 (in) : Bus_2		In2 (IN) : Bus: Bus_2	Parameter node has a coresponding Simulink port.
	Out1 (out) : Bus_1		Out1 (OUT) : Bus: Bus_1	Parameter node has a coresponding Simulink port.
	Out2 (out) : Bus_1		Out2 (OUT) : Bus: Bus_1	Parameter node has a coresponding Simulink port.
3	OBJECT FLOWS (Total = 11)		LINES (Total = 11)	Object flows vs. lines
	bc1.out1 -> mc.cIn1		bc1 -> mc/cIn1	Object flow has a corresponding Simulink line (signal/bus).
	bs1.out1 -> g1.in1		bs1 -> g1	Object flow has a corresponding Simulink line (signal/bus).
	bs1.out2 -> logic1.in1		bs1 -> logic1	Object flow has a corresponding Simulink line (signal/bus).
	g1.out1 -> bc1.in1		g1 -> bc1	Object flow has a corresponding Simulink line (signal/bus).
	In1 -> ma.aIn1		In1 -> ma/aIn1	Object flow has a corresponding Simulink line (signal/bus).
	In2 -> mb.bIn1		In2 -> mb/bIn1	Object flow has a corresponding Simulink line (signal/bus).
	logic1.out1 -> bc1.in2		logic1 -> bc1	Object flow has a corresponding Simulink line (signal/bus).
	ma.aOut1 -> bs1.in1		ma/aOut1 -> bs1	Object flow has a corresponding Simulink line (signal/bus).
	mb.bOut1 -> mc.cIn2		mb/bOut1 -> mc/cIn2	Object flow has a corresponding Simulink line (signal/bus).
	mc.cOut1 -> Out1		mc/cOut1 -> Out1	Object flow has a corresponding Simulink line (signal/bus).
	mc.cOut2 -> Out2		mc/cOut2 -> Out2	Object flow has a corresponding Simulink line (signal/bus).
6...	PartB1_libblock		PartB1_libblock	SysML block vs. Simulink model
1	PROPERTIES (Total = 3)		BLOCKS (Total = 3)	Properties vs. blocks
	bc2 : BusCreator		bc2 : BusCreator	Property has a correspoding Simulink block.
	bs2 : BusSelector		bs2 : BusSelector	Property has a correspoding Simulink block.
	p1 : Product		p1 : Product	Property has a correspoding Simulink block.
2	FLOW PORTS (Total = 2)		PORTS (Total = 2)	Flow/full/proxy ports vs. Simulink ports
	in1 (IN) : Bus_2		in1 (IN) : Bus: Bus_2	Flow/full/proxy port has a coresponding Simulink port.
	out1 (OUT) : Bus_2		out1 (OUT) : Bus: Bus_2	Flow/full/proxy port has a coresponding Simulink port.
3	CONNECTORS (Total = 5)		LINES (Total = 5)	Connectors vs. lines
	bc2.out1 -> out1		bc2 -> out1	Connector has a corresponding Simulink line (signal/bus).
	bs2.out2 -> bc2.in2		bs2 -> bc2	Connector has a corresponding Simulink line (signal/bus).
	bs2.out3 -> p1.in2		bs2 -> p1	Connector has a corresponding Simulink line (signal/bus).
	in1 -> bs2.in1		in1 -> bs2	Connector has a corresponding Simulink line (signal/bus).
	p1.out1 -> bc2.in1		p1 -> bc2	Connector has a corresponding Simulink line (signal/bus).

Figure 17 Compare SysML and Simulink after generating executable Simulink models from SysML activity & block structures

Next we will make changes on either side and use the Syndeia compare capabilities to show the differences across each connection. Syndeia is able to catch changes made on one side or the other, or changes made to both sides simultaneously, including:

- SysML side changes
    - Addition/removal of part properties in a block OR call behavior actions in an activity
    - Changing name/type of part properties in a block OR call behavior actions in an activity
    - Changing name of block used as part property OR activity used as call behavior action
    - Addition/removal of flow ports in a block OR activity parameter nodes in an activity
    - Changing name/type of ports in a block OR activity parameter nodes in an activity
    - Addition/removal of connectors in a block OR object flows in an activity
    - Rewiring of connectors in a block OR object flows in an activity
- Syndeia side changes
    - Addition/removal of model reference blocks
    - Changing name/type of model reference blocks
    - Addition/removal of inports and outports
    - Changing name/type of inports and outports
    - Changing elements of bus objects typing inports and outports
    - Addition/removal of lines (block connections)
    - Rewiring of lines

In this note, we will show only a representative subset of these changes on both the SysML and Simulink side to simulate concurrent modeling efforts. For **Overall Mission libblock** we add a new activity **mb2 : Mission B libblock** to the SysML activity structure (Figure 18), and concurrently delete a library block **logic1** in Simulink (Figure 19). We also rewire the object flows in SysML to the appropriate pins on the actions, and connect the lines/connectors in Simulink to close the gap. For **PartB1_libblock** we delete the **p1** part property and all connectors between **bs2** and **bc2** in SysML (Figure 20), while concurrently in Simulink we add a new Simulink library block **logic2** and rewire the lines to connect it between the second ports of **bs1** and **bc2** (Figure 21).

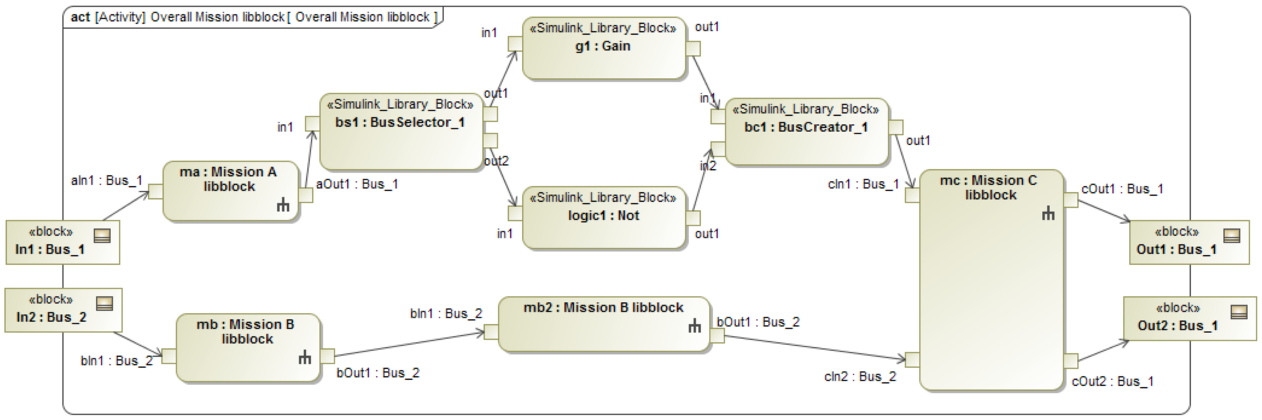


Figure 18 Added new call behavior action mb2 : Misison B libblock **& rewired object flows between pins on** mb→mb2→mc

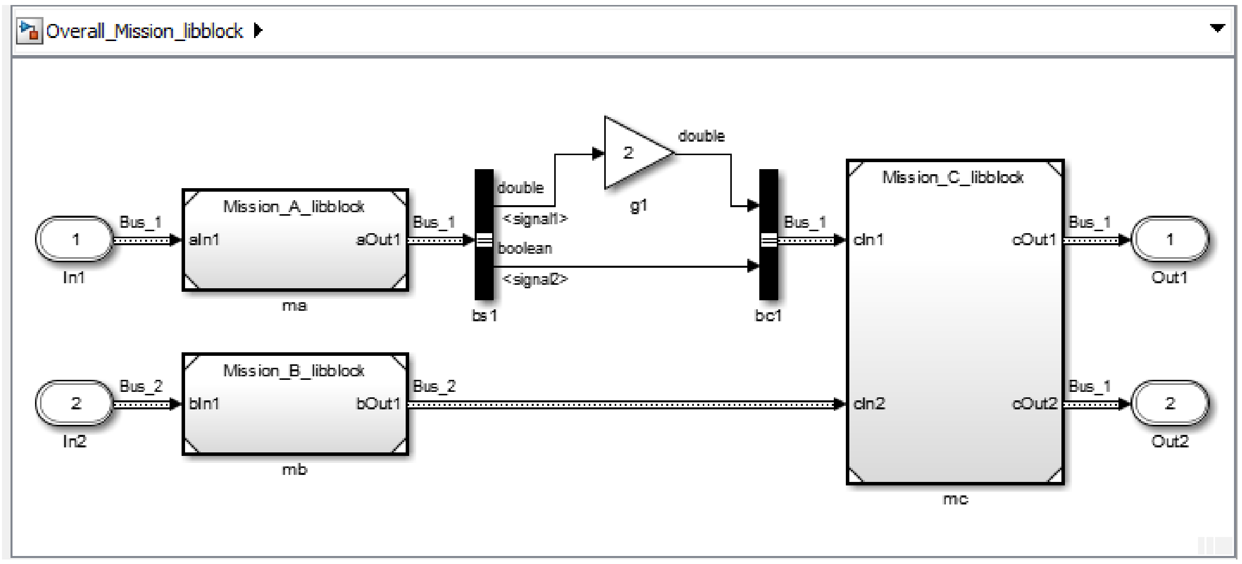


Figure 19 Deleted Simulink library block logic1 (NOT) and rewired boolean line <signal2> directly between bs1 → bc1

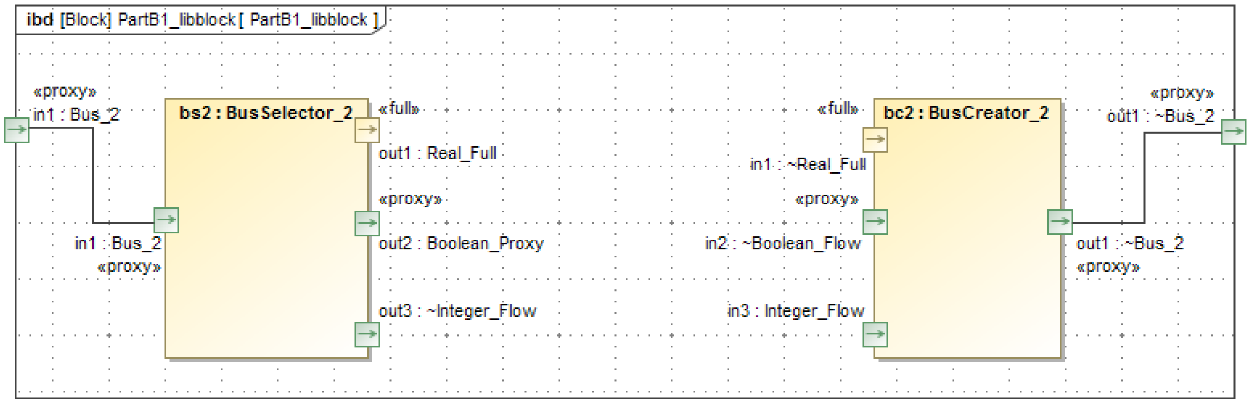


Figure 20 Deleted part property p1 and all connectors between bs2 → bc2

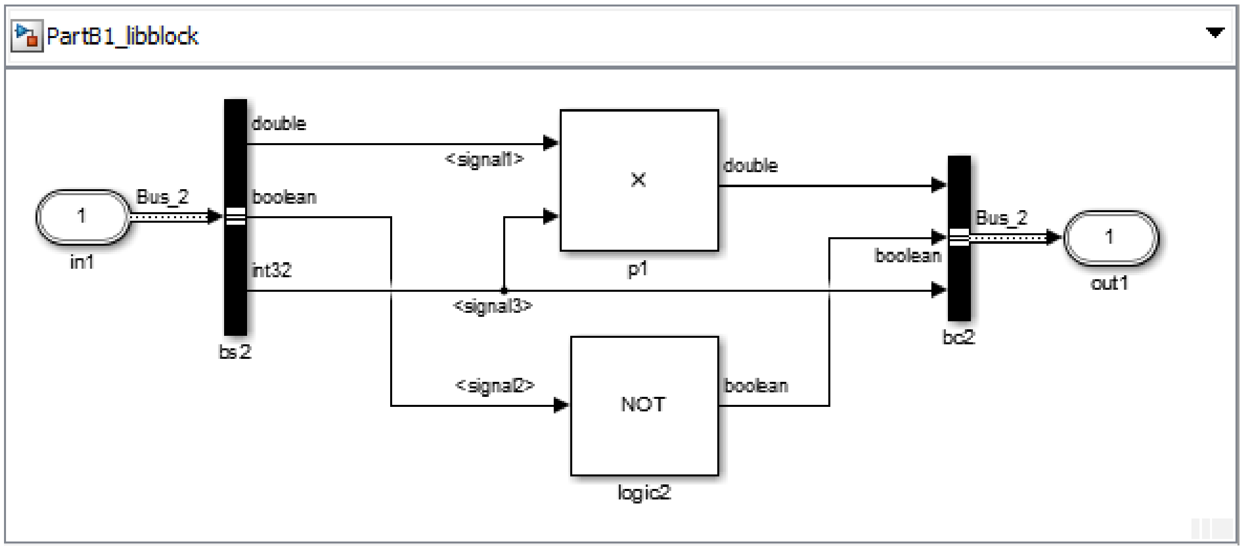
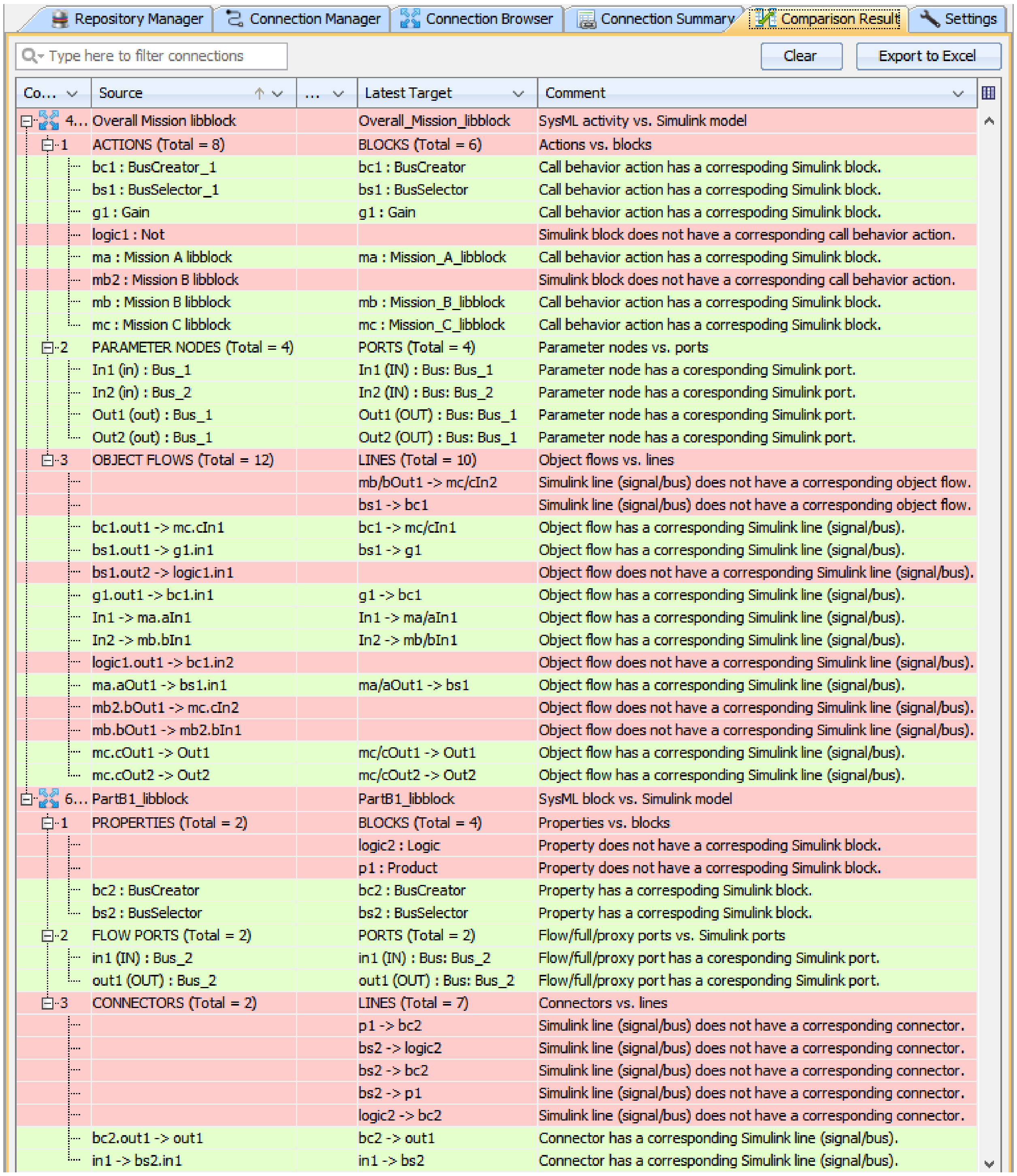


Figure 21 Added new Simulink library block logic2 (NOT) and rewired line for boolean line <signal2> from bs2 → logic2 → bc2

Then, we compare again (results shown in Figure 22).We can see that new queries have been made to both the SysML and Simulink models as changes were detected, and we can see exactly where the two models are out of sync, and which side may need to be updated, if any. Again, these are only a representative sample of the many types of changes which can be made to either side and caught by Syndeia compare.



Co…	Source	…	Latest Target	Comment
4…	Overall Mission libblock		Overall_Mission_libblock	SysML activity vs. Simulink model
1	ACTIONS (Total = 8)		BLOCKS (Total = 6)	Actions vs. blocks
	bc1 : BusCreator_1		bc1 : BusCreator	Call behavior action has a correspoding Simulink block.
	bs1 : BusSelector_1		bs1 : BusSelector	Call behavior action has a correspoding Simulink block.
	g1 : Gain		g1 : Gain	Call behavior action has a correspoding Simulink block.
	logic1 : Not			Simulink block does not have a corresponding call behavior action.
	ma : Mission A libblock		ma : Mission_A_libblock	Call behavior action has a correspoding Simulink block.
	mb2 : Mission B libblock			Simulink block does not have a corresponding call behavior action.
	mb : Mission B libblock		mb : Mission_B_libblock	Call behavior action has a correspoding Simulink block.
	mc : Mission C libblock		mc : Mission_C_libblock	Call behavior action has a correspoding Simulink block.
2	PARAMETER NODES (Total = 4)		PORTS (Total = 4)	Parameter nodes vs. ports
	In1 (in) : Bus_1		In1 (IN) : Bus: Bus_1	Parameter node has a coresponding Simulink port.
	In2 (in) : Bus_2		In2 (IN) : Bus: Bus_2	Parameter node has a coresponding Simulink port.
	Out1 (out) : Bus_1		Out1 (OUT) : Bus: Bus_1	Parameter node has a coresponding Simulink port.
	Out2 (out) : Bus_1		Out2 (OUT) : Bus: Bus_1	Parameter node has a coresponding Simulink port.
3	OBJECT FLOWS (Total = 12)		LINES (Total = 10)	Object flows vs. lines
			mb/bOut1 -> mc/cIn2	Simulink line (signal/bus) does not have a corresponding object flow.
			bs1 -> bc1	Simulink line (signal/bus) does not have a corresponding object flow.
	bc1.out1 -> mc.cIn1		bc1 -> mc/cIn1	Object flow has a corresponding Simulink line (signal/bus).
	bs1.out1 -> g1.in1		bs1 -> g1	Object flow has a corresponding Simulink line (signal/bus).
	bs1.out2 -> logic1.in1			Object flow does not have a corresponding Simulink line (signal/bus).
	g1.out1 -> bc1.in1		g1 -> bc1	Object flow has a corresponding Simulink line (signal/bus).
	In1 -> ma.aIn1		In1 -> ma/aIn1	Object flow has a corresponding Simulink line (signal/bus).
	In2 -> mb.bIn1		In2 -> mb/bIn1	Object flow has a corresponding Simulink line (signal/bus).
	logic1.out1 -> bc1.in2			Object flow does not have a corresponding Simulink line (signal/bus).
	ma.aOut1 -> bs1.in1		ma/aOut1 -> bs1	Object flow has a corresponding Simulink line (signal/bus).
	mb2.bOut1 -> mc.cIn2			Object flow does not have a corresponding Simulink line (signal/bus).
	mb.bOut1 -> mb2.bIn1			Object flow does not have a corresponding Simulink line (signal/bus).
	mc.cOut1 -> Out1		mc/cOut1 -> Out1	Object flow has a corresponding Simulink line (signal/bus).
	mc.cOut2 -> Out2		mc/cOut2 -> Out2	Object flow has a corresponding Simulink line (signal/bus).
6…	PartB1_libblock		PartB1_libblock	SysML block vs. Simulink model
1	PROPERTIES (Total = 2)		BLOCKS (Total = 4)	Properties vs. blocks
			logic2 : Logic	Property does not have a correspoding Simulink block.
			p1 : Product	Property does not have a correspoding Simulink block.
	bc2 : BusCreator		bc2 : BusCreator	Property has a correspoding Simulink block.
	bs2 : BusSelector		bs2 : BusSelector	Property has a correspoding Simulink block.
2	FLOW PORTS (Total = 2)		PORTS (Total = 2)	Flow/full/proxy ports vs. Simulink ports
	in1 (IN) : Bus_2		in1 (IN) : Bus: Bus_2	Flow/full/proxy port has a coresponding Simulink port.
	out1 (OUT) : Bus_2		out1 (OUT) : Bus: Bus_2	Flow/full/proxy port has a coresponding Simulink port.
3	CONNECTORS (Total = 2)		LINES (Total = 7)	Connectors vs. lines
			p1 -> bc2	Simulink line (signal/bus) does not have a corresponding connector.
			bs2 -> logic2	Simulink line (signal/bus) does not have a corresponding connector.
			bs2 -> bc2	Simulink line (signal/bus) does not have a corresponding connector.
			bs2 -> p1	Simulink line (signal/bus) does not have a corresponding connector.
			logic2 -> bc2	Simulink line (signal/bus) does not have a corresponding connector.
	bc2.out1 -> out1		bc2 -> out1	Connector has a corresponding Simulink line (signal/bus).
	in1 -> bs2.in1		in1 -> bs2	Connector has a corresponding Simulink line (signal/bus).

Figure 22 Compare SysML and Simulink after making changes shown in Figures 18-21

Summary

The intent of this second note in our series has been to illustrate the possibilities for using a SysML model to generate an executable Simulink model with native Simulink library blocks using Syndeia, so that the resulting model is ready to be executed in the MATLAB/Simulink environment using an appropriate script. As in the previous installment, we showed that Syndeia also preserves and version manages connections between Simulink models and SysML blocks or activities, which can then be used to compare and catch changes that have been made to either side.

To explore the use case where SysML block and activity structures are used to generate a skeletal structure of Simulink models either as an internal block structure with part properties, or an activity structure with call behavior actions, see the previous Part 1. The following Part 3 will continue exploring the Syndeia Simulink interface for the reverse case of generating SysML activity and block structures from a Simulink model reference structure. Further use cases are in the works for Syndeia 3.0, so stay tuned for Part 4 and beyond!

If you are interested in trying Syndeia, follow the instructions here to get a free 30-day evaluation license and download instructions: http://intercax.com/products/syndeia/download/

About the Author

Rose Yntema (rose.yntema@intercax.com) is Applications Engineer for Intercax LLC, Atlanta, GA.
For further information, visit us at www.intercax.com or contact us at info@intercax.com.