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 1 – SysML → Simulink Skeletal 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. We are

introducing a new 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 will highlight various capabilities of Syndeia for using Simulink with SysML to coordinate the simulation-based design process of a system. Over the next few installments we will outline different scenarios for using Syndeia 2.0 to generate, connect, and compare Simulink and SysML models. Part 1 will show how SysML block and activity structures can be used to generate basic 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 will 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 first installment of the series, we 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. This skeletal model structure, with ports and interfaces defined by the System Engineer in SysML and generated into Simulink, can then be used by a domain engineer with Simulink expertise to flesh out the functional design of the simulation model. Meanwhile, persistent connections between SysML elements and Simulink models have been created by Syndeia, so that as changes are made to either side, we can compare across those connections to show what changes have been made and whether the ports and interfaces remain in sync.

Generating Simulink models with atomic signal flows

The first scenario we will explore for generating a skeletal Simulink model from a SysML model involves only atomic signal flows, represented by SysML value types *Real*, *Integer*, or *Boolean*. This means that a single signal can flow through a port or activity parameter node, which will map to inports and outports in Simulink which likewise only allow a single signal of comparable type to flow:

- *Real → double*
- *Integer → int32*
- *Boolean → boolean*

A skeletal structure can be defined in SysML by using either an internal block structure or an activity structure. We begin here in Figure 1 with an internal block diagram (IBD) to represent an internal block structure in SysML. The diagram frame indicates the top-level block in the structure, *System*. Other blocks, such as *PartA*, have been used as part properties, i.e. *pA : PartA*. Further structure is then nested inside each block with its own part properties as needed in separate IBDs, such as *PartA*, shown in Figure 2. As the notes on the different ports in Figure 1 indicate, we have used multiple port options in SysML (flow ports, full ports, and proxy ports) to show the various possibilities that are both

allowed by SysML and supported by the tool interface, but the choice of which port type to use may be influenced by other factors. Ports with compatible directions may be joined by connectors as shown – when connectors join two ports at the same level, such as *pA.out1* → *pc.in1*, the directions (out/in) are opposite, while connectors joining ports at nested levels, such as *in1* → *pA.in1*, have the same direction.

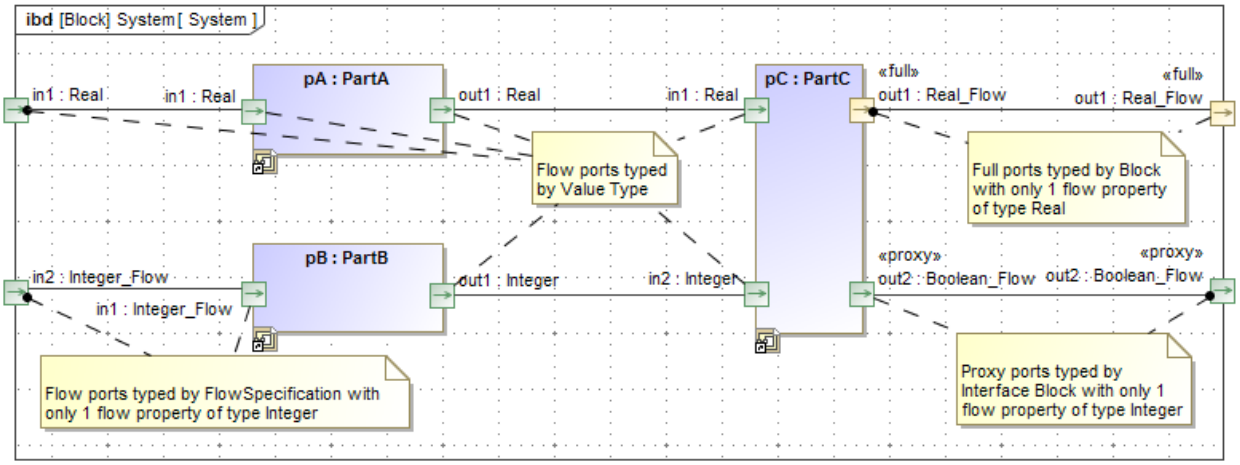

Figure 1 IBD of *System* internal block structure with comments indicating different ports and how they are typed

Figure 3 below shows the structure in the containment tree of the *System* block from the IBD in Figure 1, and Figure 4 shows the *Boolean_Flow* Interface Block, *Real_Flow* Block, and *Integer_Flow* FlowSpecification used to type the proxy port *out2*, the full port *out1*, and the flow port *in2*, respectively (the flow port *in1* is typed by the *Real* value type from the SysML profile). The flow port concept is being effectively replaced by the proxy port and full port concepts in SysML 1.3, but older models will still be supported, so we show them here.

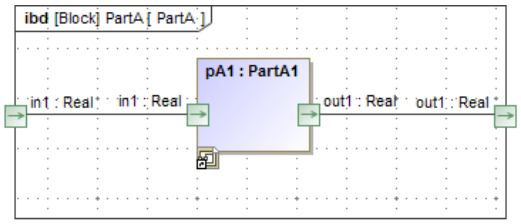

Figure 2 *PartA* IBD nested inside *System* IBD where *PartA* is used as part property *pA : PartA*

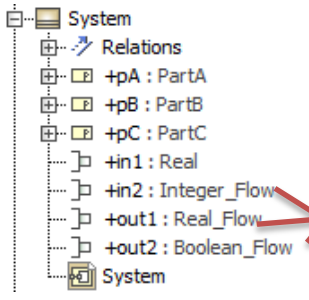

Figure 3 *System* block with various ports and port types

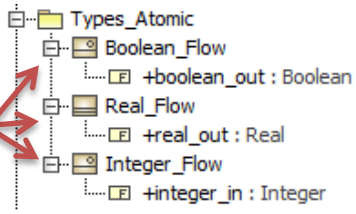

Figure 4 Flow properties in Interface Block, Block, and FlowSpecification

Activity structures can follow a similar pattern, as seen in Figure 5 below. The diagram frame represents the top-level activity, *Overall Mission*, while other activities, such as *Mission A*, have been used as call behavior actions, i.e. *ma : Mission A*. Again, further activities may be used as call behavior actions inside of these blocks, which have nested activity diagrams of their own. In this case, activity parameter nodes on the frame (i.e. *In1 : Real*) are connected by object flows to pins in the same direction (i.e. *aIn1 : Real*) on the call behavior actions (i.e. *ma : Mission A*), or two pins of opposite directions are connected at the same level (i.e. *ma.aOut1* → *mc.cIn1*). In this basic model the activity

parameter nodes are typed by SysML value types, i.e. *Real*, *Integer*, or *Boolean*. Pins on call behavior actions correspond to the activity parameter nodes of the referenced activity.

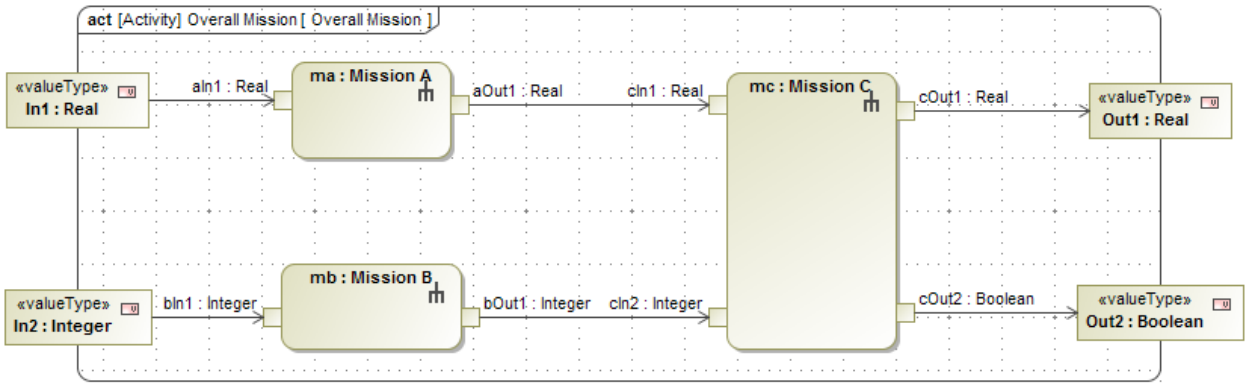

Figure 5 Activity structure of *Overall Mission* for generation in Simulink

In the Syndeia dashboard, Connection Manager tab, shown below, the *Overall Mission* activity (left) can be dragged to an empty folder in a local file system repository as shown in Figure 6 to generate a Simulink model structure.

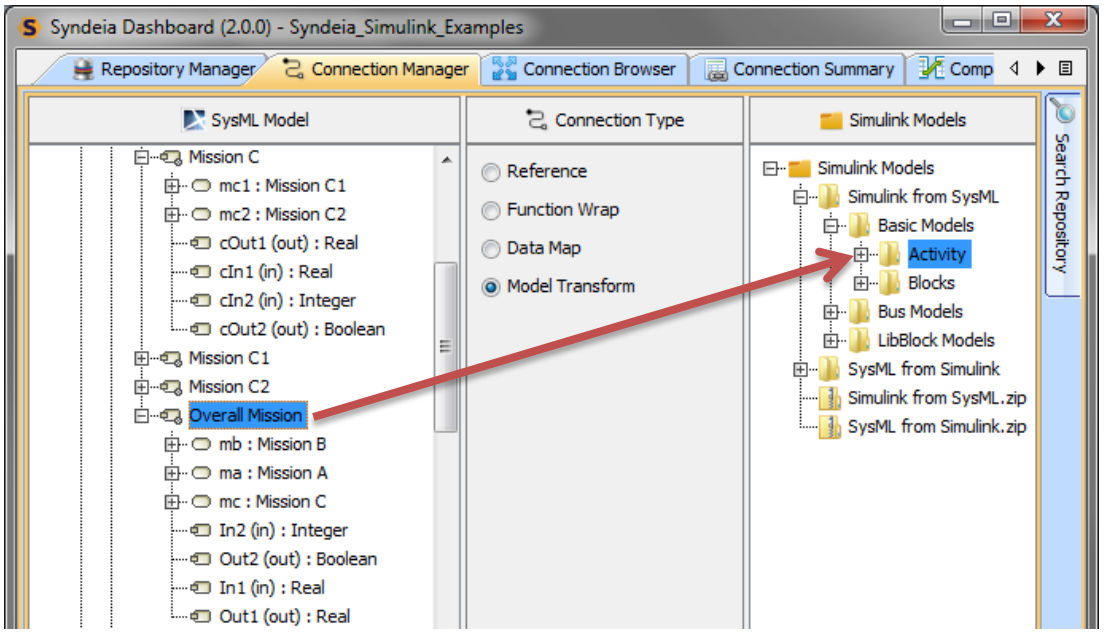

Figure 6 Creating Simulink model structure from activities in SysML

As we can see in Figure 7 SysML structure on the left and Simulink model structure generated from SysML on the right on the right, Syndeia has generated multiple Simulink model files, one for each activity. The file extension will be the default set in local MATLAB, in this case it is *slx*, but for older versions of MATLAB, models with the *mdl* extension will be created, or for newer versions if the user's default setting in MATLAB has been changed to this. If we expand the file, queries are made to get the structure of the model inside, highlighted here, which has the same name as the file without extension.

Just as an activity is mapped to a Simulink model, call behavior actions are mapped to model reference blocks in Simulink, i.e. *mc : Mission_C*, represented here by a white box. This has been expanded in the figure to show the nested structure, but it is itself a reference to the *Mission_C* model in another file, *Mission_C.slx*, mapped to the *Mission C* activity on the left. Likewise, input and output parameter nodes/pins have been mapped to inports and outports in Simulink, represented by arrows either into or out of white boxes. Object flows have been mapped to signals/lines in Simulink, represented by plain arrows.

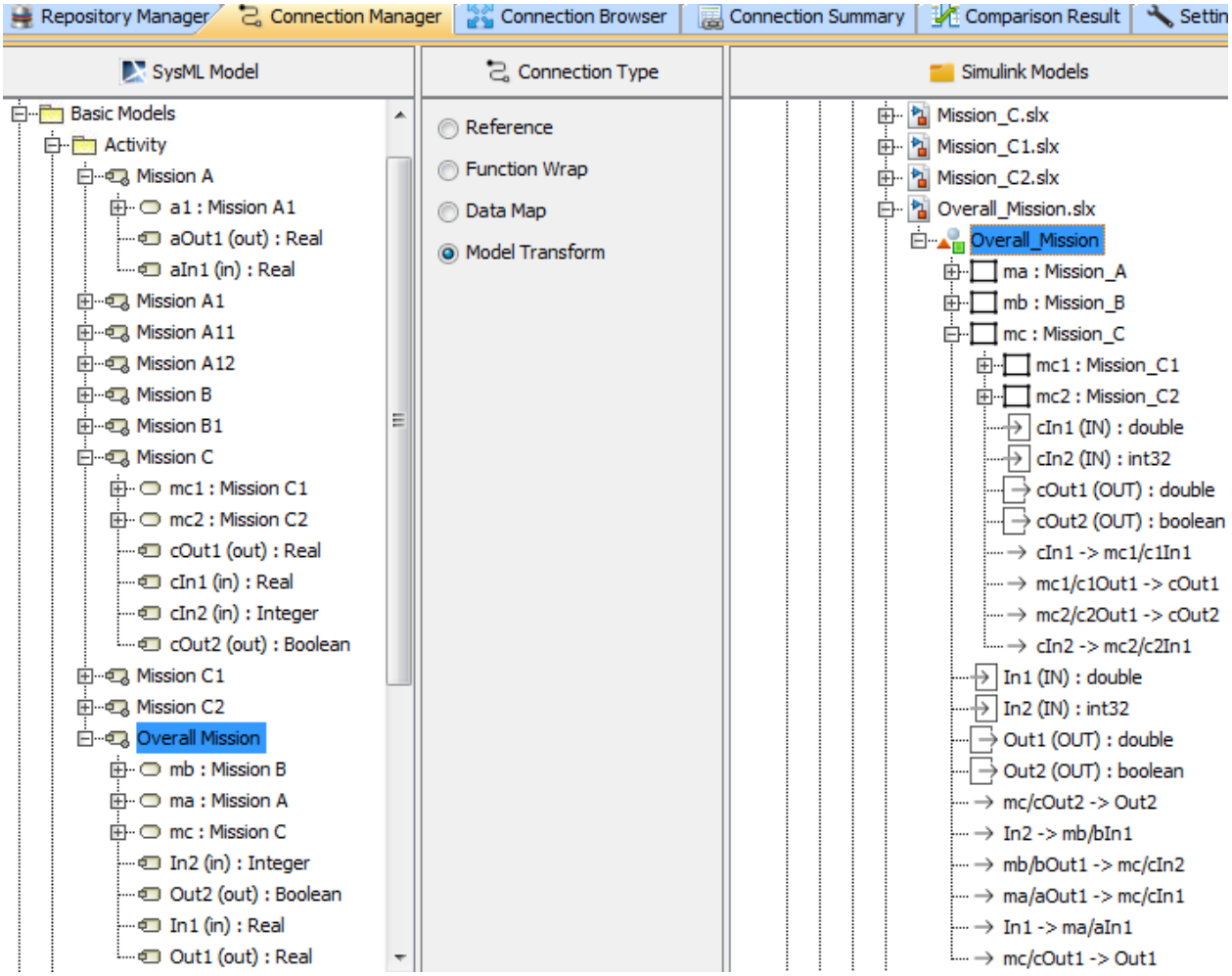

Figure 7 SysML structure on the left and Simulink model structure generated from SysML on the right

Similarly, the block structure from Figure 1-Figure 4 may be used to generate a Simulink model structure, with *PartA* shown expanded to multiple levels in Figure 8 below after the *System* block was dragged to a folder as indicated by the arrow. As with the activity structure, blocks map to models in Simulink (contained in model files of the same name), and SysML part properties map to model reference properties in Simulink. For flow ports, full ports, and proxy ports as discussed above, where they are either typed by SysML value types in the case of flow ports, or where their typing block, interface block, or FlowSpecification has only one flow property typed by SysML value types making it an

atomic port, the types are mapped to data types in Simulink as explained above. Connectors between ports in SysML map to Simulink lines/connectors as with object flows above.

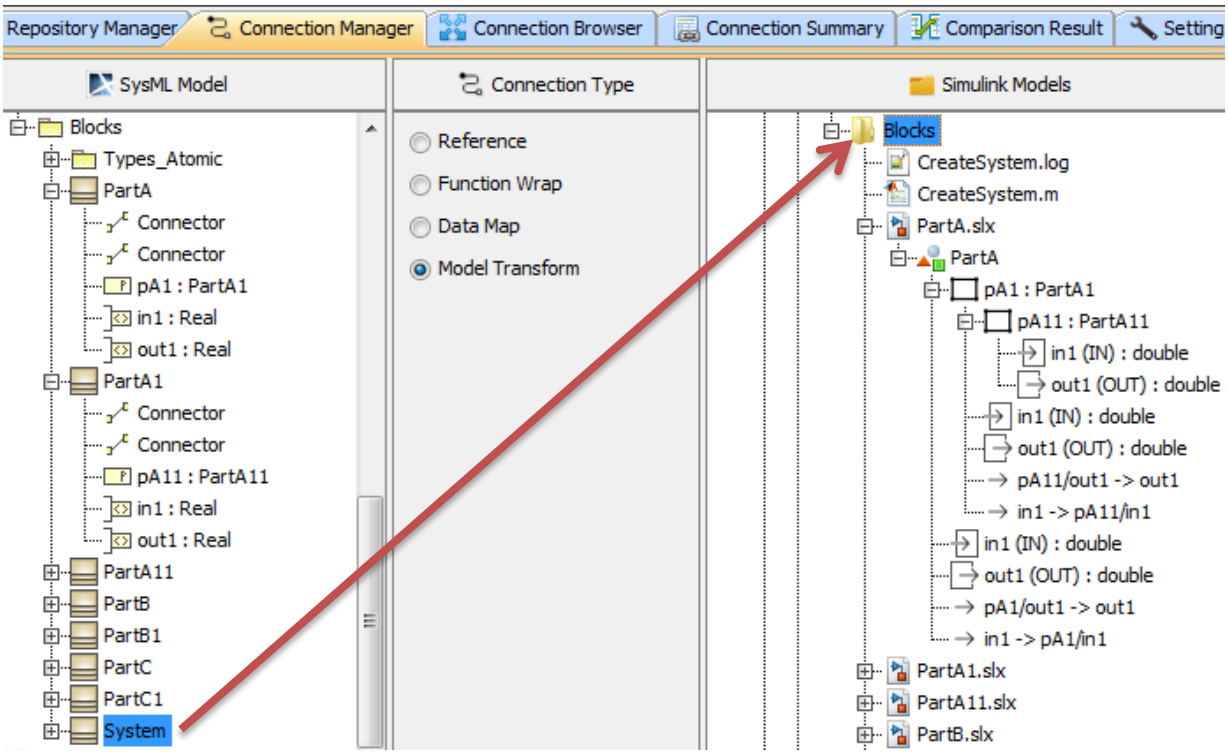

Figure 8 Generating Simulink models from SysML internal block structures

We can open the newly-generated Simulink models from the Syndeia dashboard to view them in the native MATLAB environment as shown in the figures below, and arrange the model elements for visual clarity if desired.

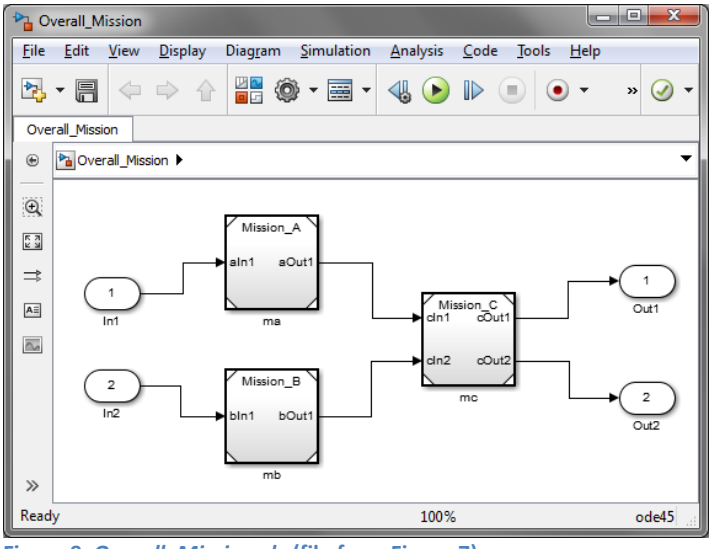

Figure 9 *Overall_Mission.slx* (file from Figure 7)

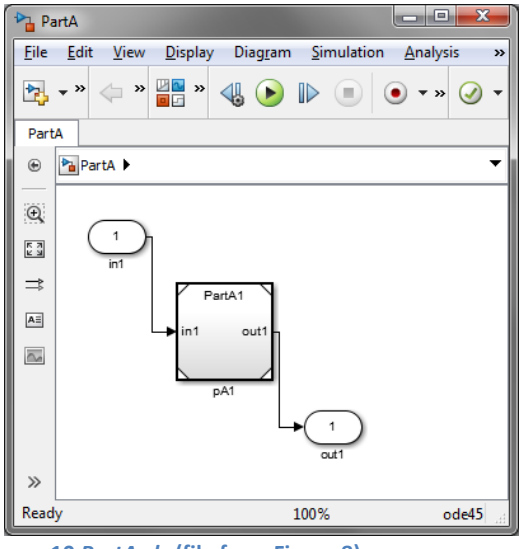

Figure 10 *PartA.slx* (file from Figure 8)

Generating Simulink models with signal buses

The next level of complexity that we will explore in Part 1 is to group multiple signals into buses in order to allow multiple signals on connectors and ports (or object flows and activity parameter nodes). These will be used to generate Simulink models with bus objects typing the inports and outports. For internal block structures, this can be done by adding multiple flow properties to the element that types the port (i.e. Interface Block for proxy port, Block for full port, or FlowSpecification for flow port as described above for the atomic signal case in Figure 4). Examples of each with multiple flow properties are shown below in Figure 11. For the equivalent functionality in activity structures, multiple flow properties should be added to blocks, as in Figure 12, which are used to type the activity parameter nodes, as shown in Figure 13.

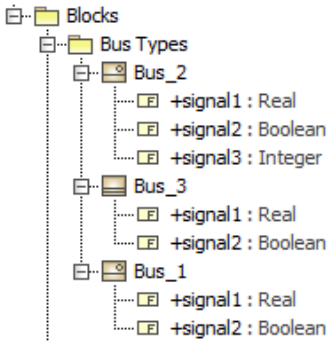

Figure 11 Port typing elements with multiple flow properties

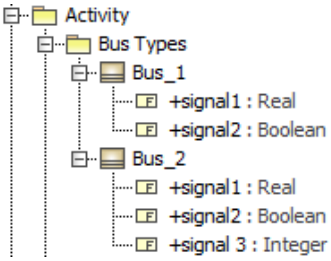

Figure 12 Parameter node typing elements with multiple flow properties

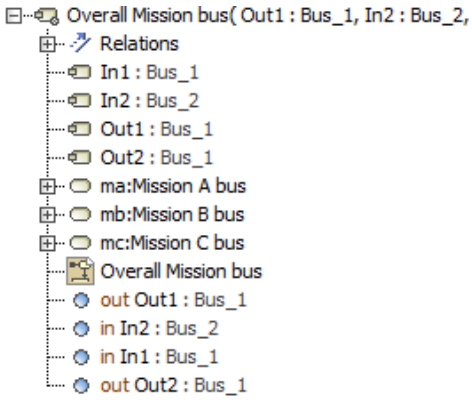

Figure 13 *Overall Mission* activity with call behavior actions and parameter nodes

As before, the top-level activity or block can be dragged in the Syndeia dashboard from SysML Model to a folder in a local file system repository to generate Simulink models. The same types of models are generated as before, but this time the bus information for each model is stored in a *.mat* file of the same name as the model. This *.mat* file of bus definitions is loaded by the *PreloadFcn* of the model. The reason a separate bus file is generated for each model that uses buses is so that the models may be used modularly from any level, not just the top level. When each model is expanded in Syndeia, the bus objects are simultaneously loaded into the MATLAB workspace, so that Simulink can show each bus port expanded with the signals and their types as shown in Figure 14 and Figure 15 below.

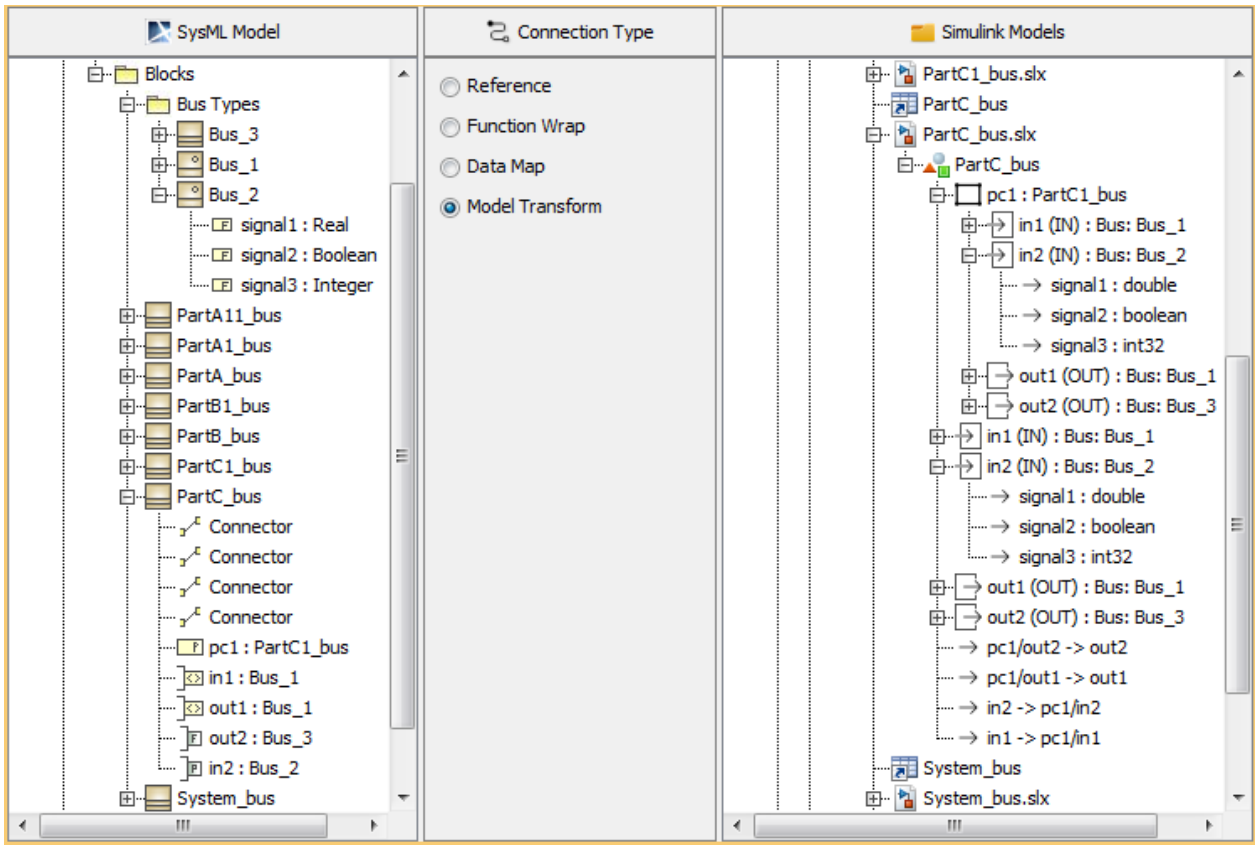

Figure 14 *PartC_bus* block and Simulink model shown side by side, with *Bus_2* expanded as well

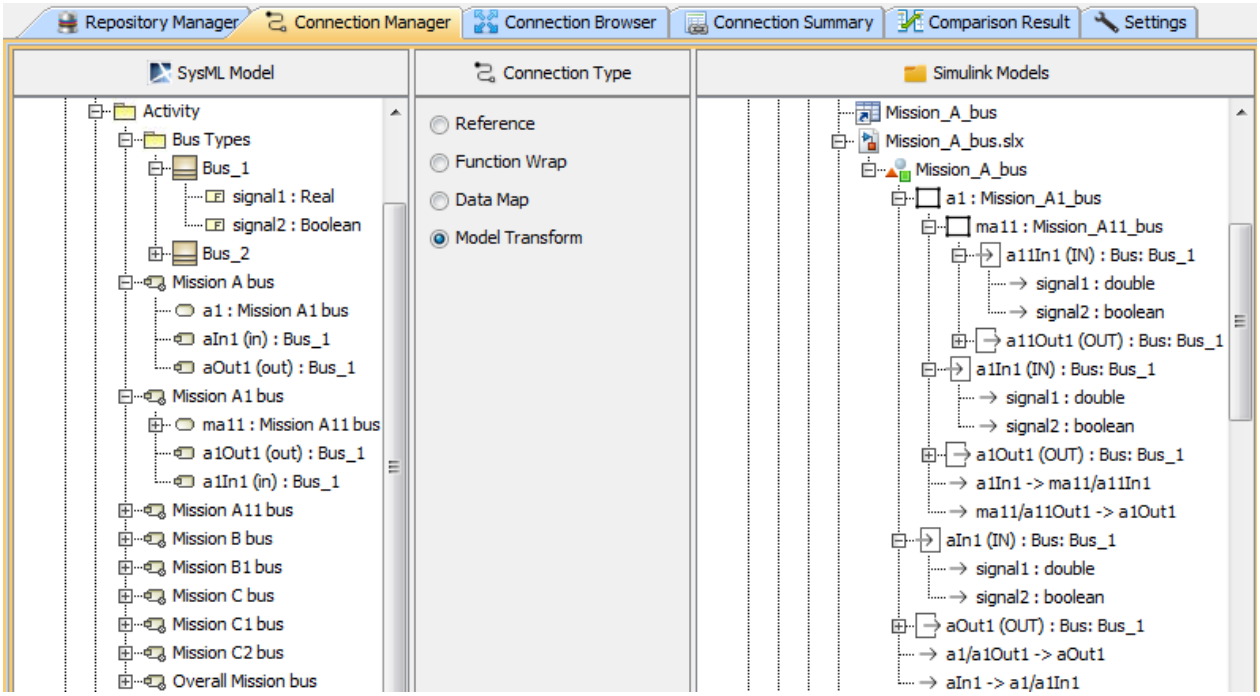

Figure 15 *Mission_A_bus.slx* shown expanded to multiple levels on the right hand side with source activities on the left

We can again open the newly-generated Simulink models from Syndeia to view and rearrange them in the native MATLAB environment as shown in the figures below. This time the ports have a double-line border to indicate that they are bus ports. In the scenarios we explore in Part 1, the lowest-level models such as *PartC1_bus* in Figure 17 have been left empty with only ports in place, so that an expert Simulink modeler can take the basic architecture built by the system engineer and fill it out with specific connections and behaviors.

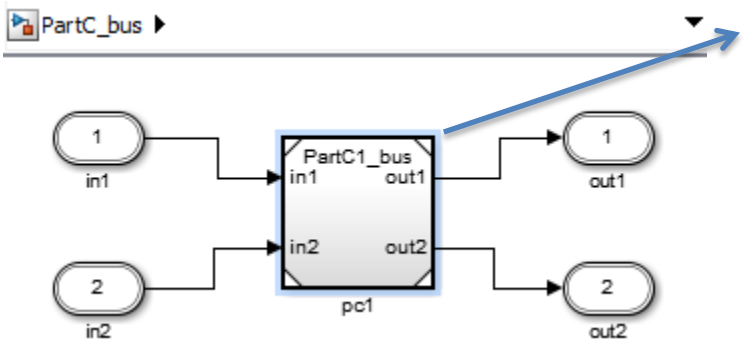

Figure 16 *PartC_bus.slx* (file from Figure 14)

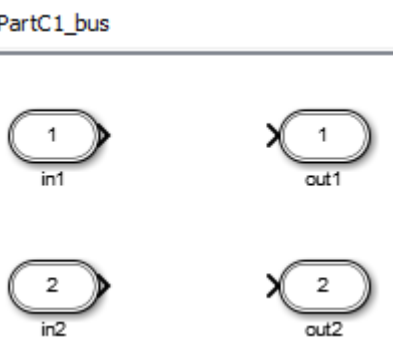

Figure 17 *PartC1_bus.slx* (referenced in Figure 16)

To see the bus object that is used to type a particular port we can double-click that port and go to the Signal Attributes tab, then expand the >> button next to the Data Type field as shown in Figure 18. Click Edit to see and/or edit the signals and their datatypes as shown in Figure 19.

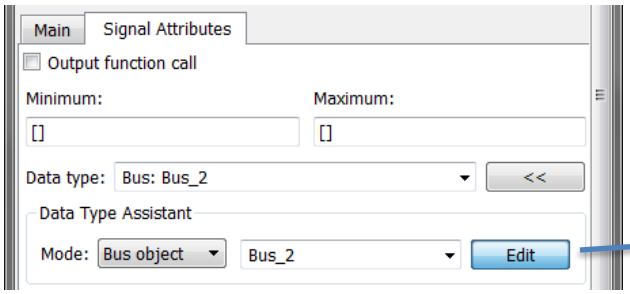

Figure 18 Signal attributes and data type for port *in2*

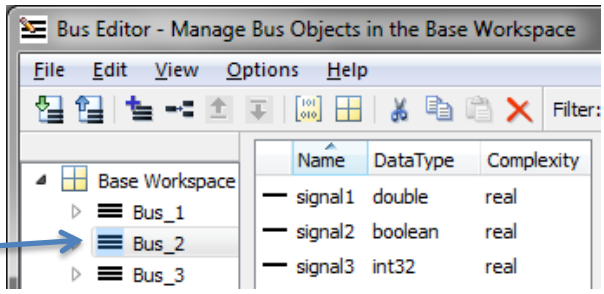

Figure 19 Base workspace and signals for *Bus_2* object

Comparing Simulink models with their SysML sources

Finally, we will utilize the connections created and persisted when Simulink models were generated from SysML blocks and activities. First, we will compare the connections before changing anything to show what happens when the models are in sync. Then 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, but first we show the comparison of the untouched models. By navigating to the Connection Browser tab we can see all of the relevant SysML elements and expand them to show both the structural elements and the connections to elements in repositories. Figure 25 shows two connections selected – one between the *Overall Mission bus* activity and the *Overall_Mission_bus* target, and another between the *PartA11_bus* block and the *PartA11_bus* target, both in the Simulink Models repository. By right-clicking after multiple-selecting these connections, we can see whether any changes have been made between the SysML elements and the target Simulink models. This is done by making calls to MATLAB to query the Simulink models and then comparing the elements returned with the elements in SysML.

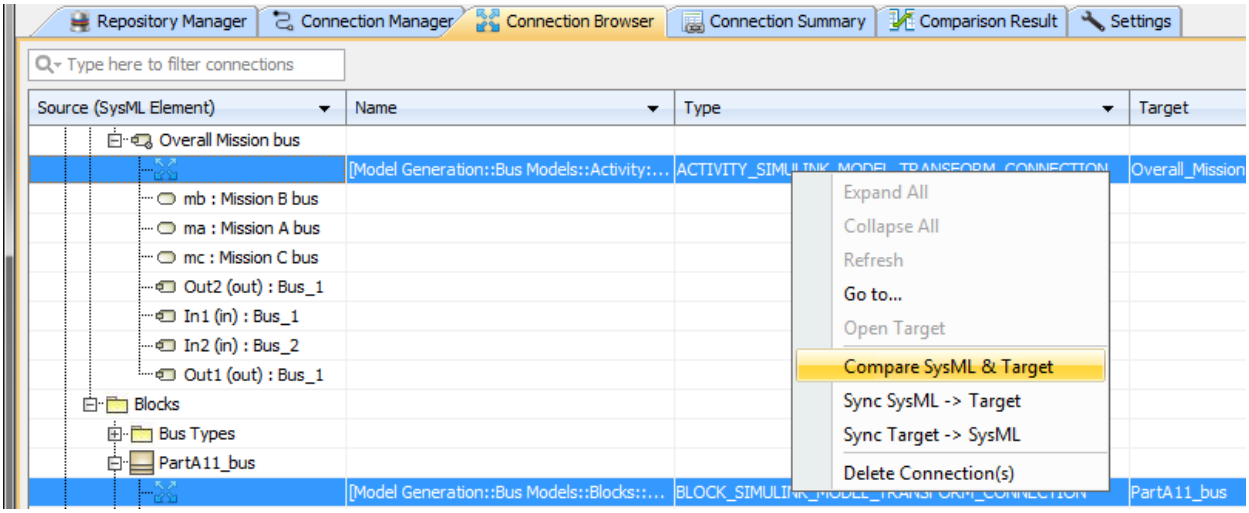

Figure 20 Compare SysML and Simulink for an activity and a block

The Comparison Result tab shown in Figure 21 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.

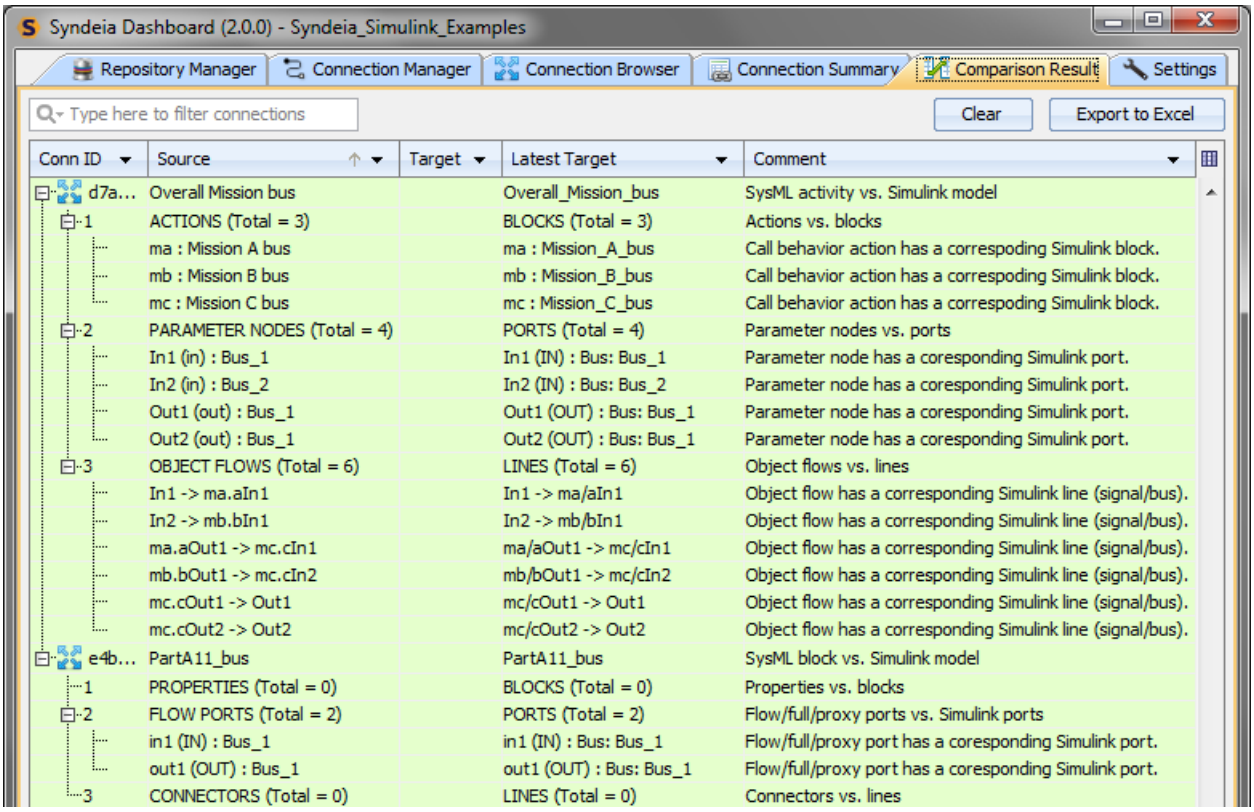

Figure 21 Compare results for connections queried from Figure 20

Then we make changes to both the SysML models and Simulink files to simulate concurrent modeling efforts. For the *Overall Mission bus* activity we may rename a call behavior action (*alpha*, Figure 22), and concurrently rename a port in Simulink (*Bus1In*, Figure 23). For the *PartA11_bus* block we may add a new port (*in2 : Real_Flow*, Figure 24), while concurrently in Simulink we add a Simulink library block and connect it with new lines (*Gain*, Figure 25). Then, we compare again (results shown in Figure 26).

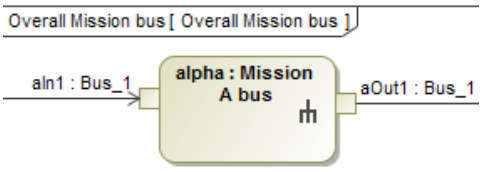

Figure 22 Changing call behavior action name

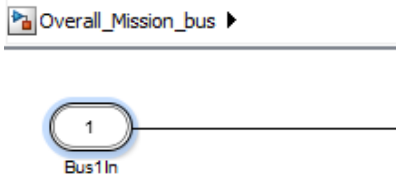

Figure 23 Changing the name of inport in Simulink

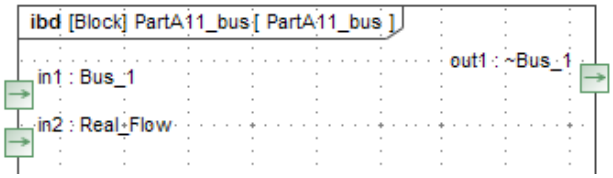

Figure 24 Adding new flow port in SysML

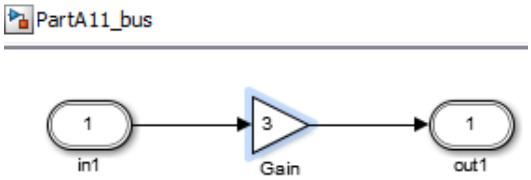

Figure 25 Adding new Simulink library block and lines

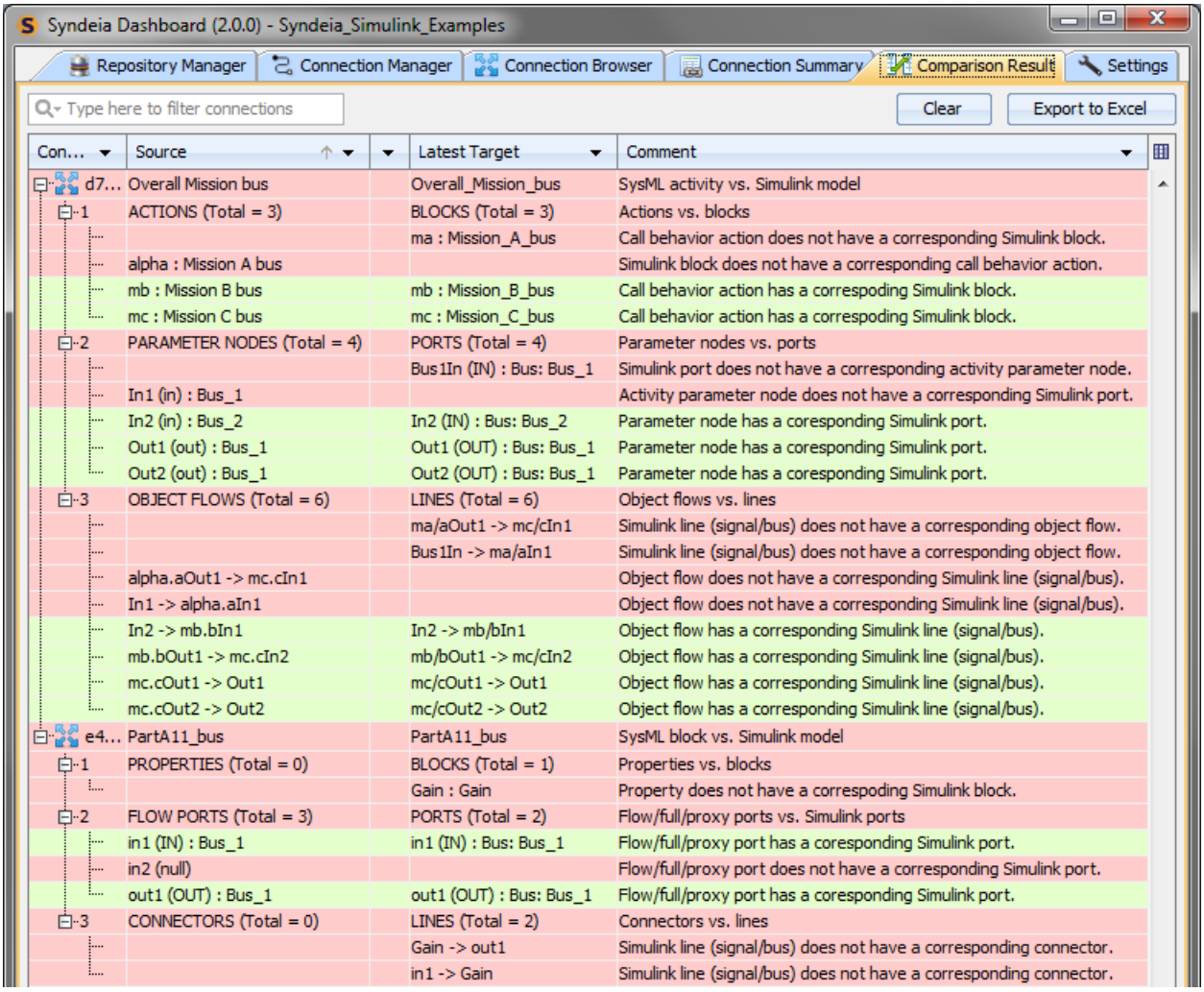

Figure 26 Compare SysML and Simulink for a row after changes made on both sides

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 needs to be updated, if any. As mentioned above, these are only a representative sample of the many types of changes which can be made to either side and caught by Syndeia compare.

Summary

The intent of this first note has been to illustrate the possibilities for using a SysML model to generate a skeletal Simulink model using Syndeia, which then preserves and version manages the connections. The connections created through model transform operations were then used to compare and catch changes made to either side, i.e. a domain engineer filling out the skeletal Simulink model into a functional simulation model, or the System Engineer making changes to structure or interfaces on the SysML side. We have shown that this process can be carried out similarly for either internal block structures with ports, part properties, and connectors, or for activity structures with activity parameter

nodes, call behavior actions, and object flows. Furthermore, the signals that flow can be modeled as atomic flows, or as buses that collect multiple flows.

The following parts 2 & 3 will continue exploring the Syndeia Simulink interface for the case of

- SysML → Simulink executable model including native Simulink library blocks
- Simulink model reference structure → SysML activity or block 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.