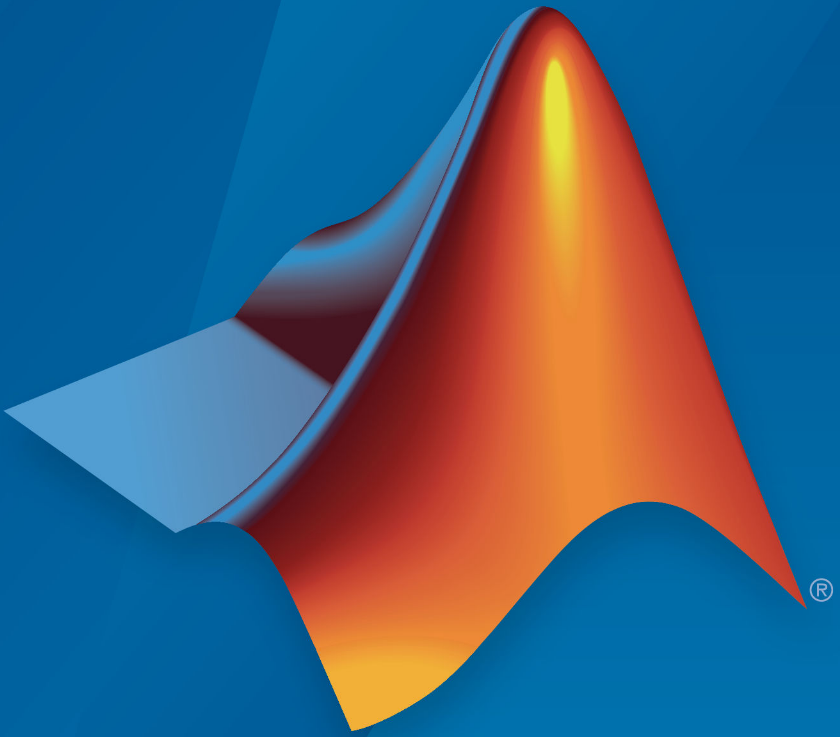
System Composer™

Getting Started Guide



MATLAB® & SIMULINK®

R2019**a**

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

System Composer™ Getting Started Guide

Revision History

March 2019	Online only	New for Version 1.0 (Release 2019a)

Contents

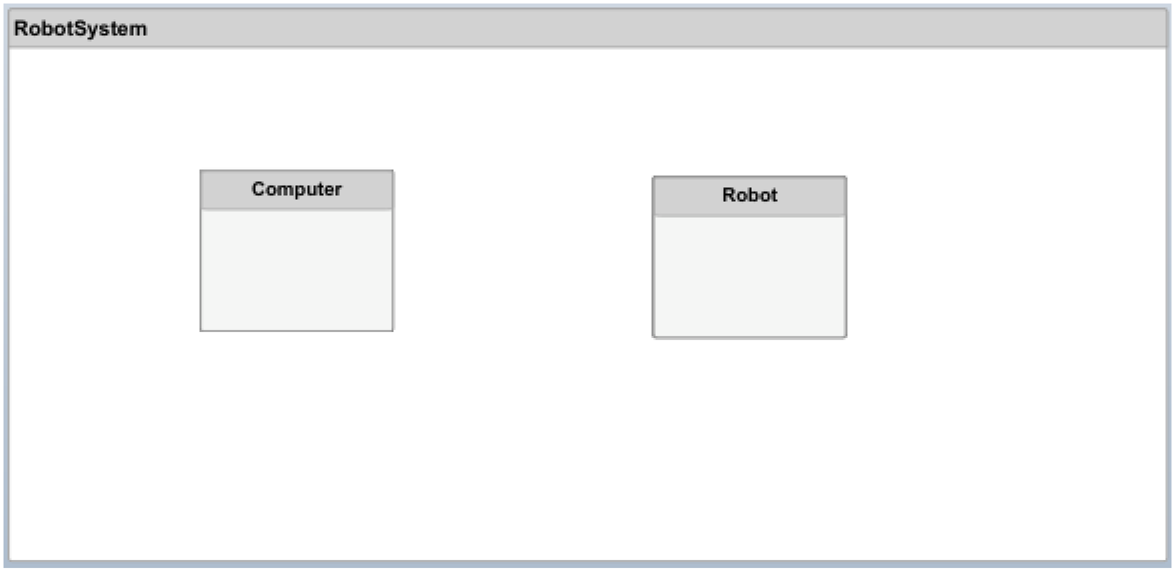
Compose an Architecture Model

Compose and Analyze a System

A system is a construct of different elements that serve a goal that cannot be achieved by any of the elements alone. The elements of a system can include mechanical parts, electrical circuits, computer hardware, and software. A system solution includes a set of elements, their characteristics, and properties. You can use System Composer to create architecture models using structural and behavioral diagrams that all act on the same underling model. This way, you ensure that a change in one diagram is reflected in the other diagrams, resulting in a consistent system model that you can communicate. With System Composer, you can:

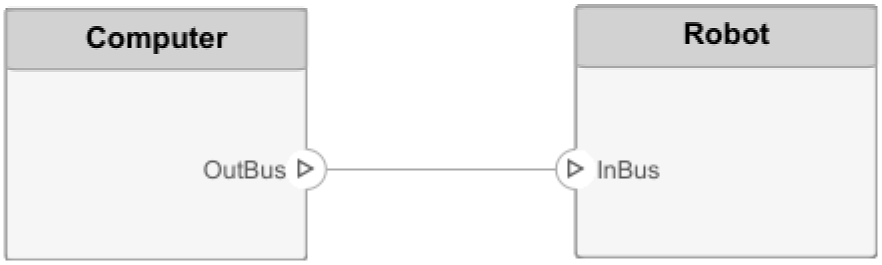
- Create structural models using hierarchical functional, logical, and physical architecture diagrams
- Support specific architectural requirements by customizing architectural types
- Validate behavior, refine, and elaborate requirements
- Perform static analysis and trade studies to optimize system architectures

Consider a mobile robotic system where the a computer sends a target location to the robot wirelessly. This system has two primary components: The computer and the robot. You represent these in System Composer with two component blocks:



You can add non-visual properties to a component to capture its specifications relevant to the problem at hand. For example, if the total power consumption of the system is a concern, a `Power Consumption` property is necessary. You can add this property to a component using an "electrical component" stereotype.

Connections are essential in describing a system as a network of components. In System Composer, you define ports on each component and connect them:



You can write an interface to fully specify a connection (and its associated ports). An interface can consist of multiple data elements with various dimensions, units, data types, etc. You can also associate interfaces with unconnected ports during component design, to enable consistency checking when connecting that port.

Requirements are integral to the system engineering process. Some are related to the functionality of the overall system, and some are non-functional. Some requirements of the overall robot system could be:

- The total power consumption
- The time between issuing the command on the computer and completing the motion of the robot arm
- Precision in positioning

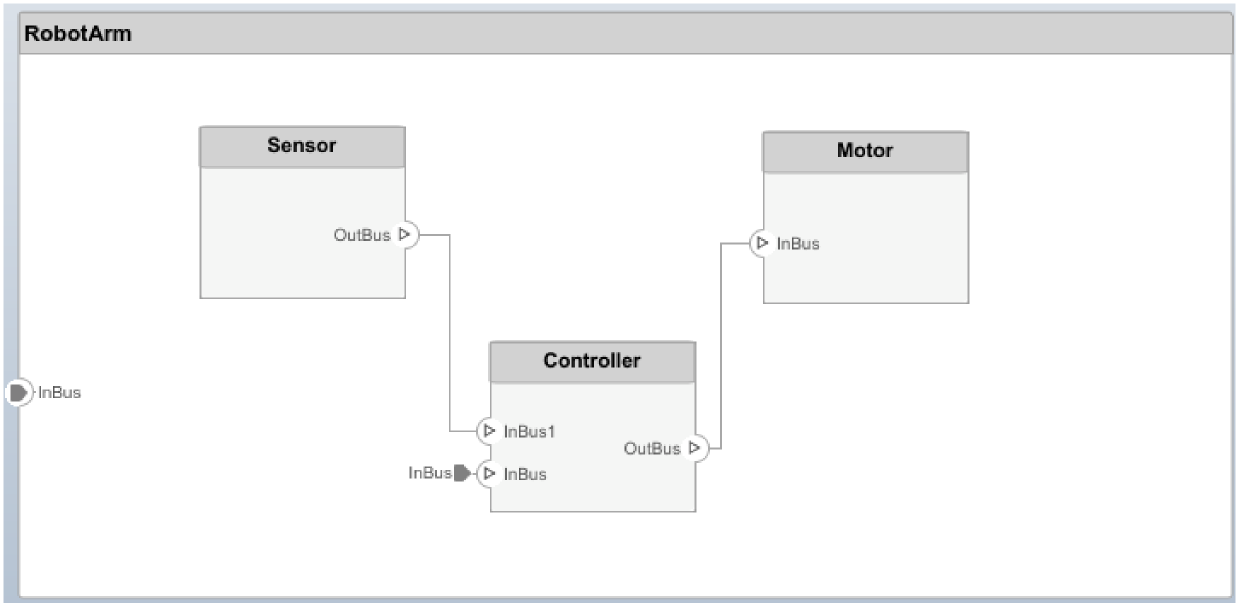
System Composer fully integrates with Simulink® Requirements™ to allocate and trace requirements with system elements.

Often, these overall requirements are broken down into requirements for each component, and these evolve during design, such as these requirements for the robot itself:

- The maximum speed

- Sensitivity of proximity sensors

These requirements point to subcomponents of the robot — motors and sensors. Represent these subcomponents by decomposing the component:



You can associate components with requirements at any level of the system.

Sometimes an overall analysis of the system is necessary to verify requirements, or to serve as requirements to design of other systems. An example would be a box to house the robot system in extreme conditions. Customizing model elements with non-functional properties such as weight or temperature sensitivity enable such analyses.

The next step in system design is designing the actual behavior of the components in Simulink. Link System Composer components to Simulink models to trace architectural design to behavioral design.

See Also

More About

- “Compose an Architecture Model” on page 1-6
- “Inspect Components in Custom Views” on page 1-26
- “Analyze Architecture Model” on page 1-30

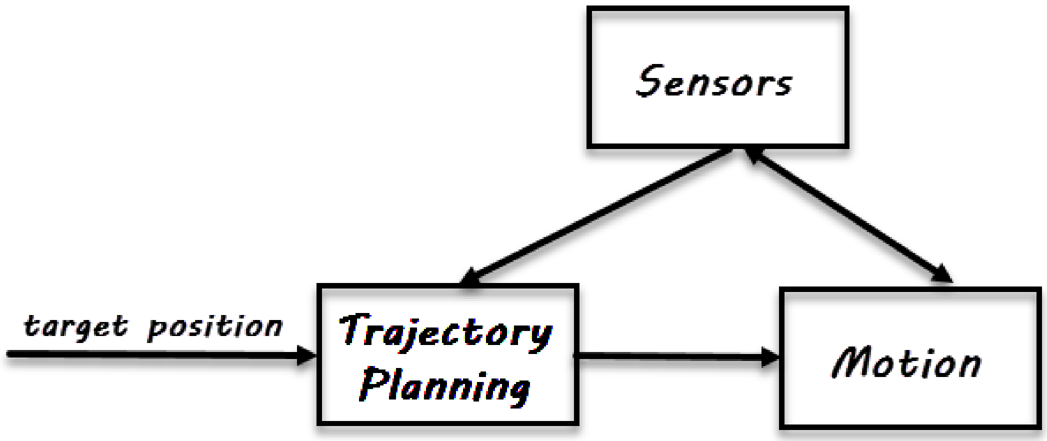
Compose an Architecture Model

In this section...

Create an architecture model of a mobile robot, that consists of sensors, motion, and a planning algorithm. Define interfaces and link requirements.

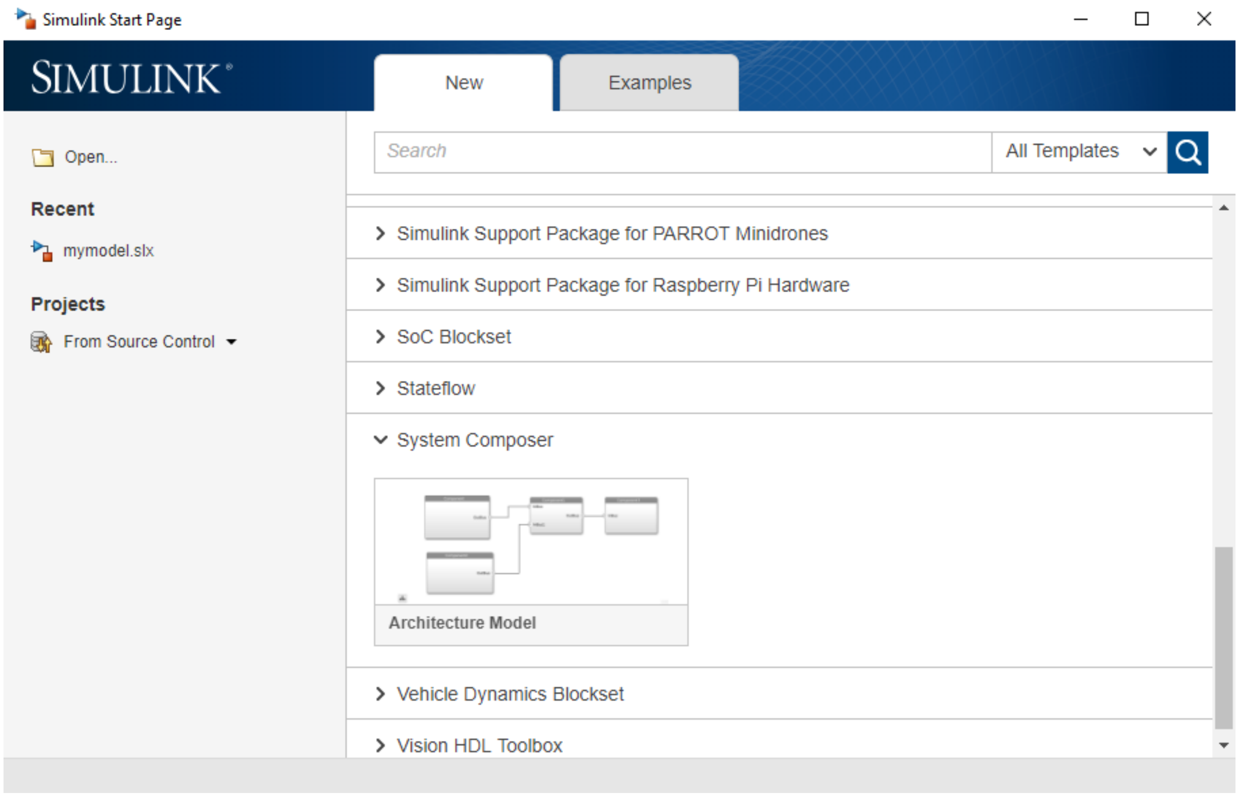
Visually Represent System Architecture

Capture the architecture of the robot arm using System Composer. The robot arm consists of these components:

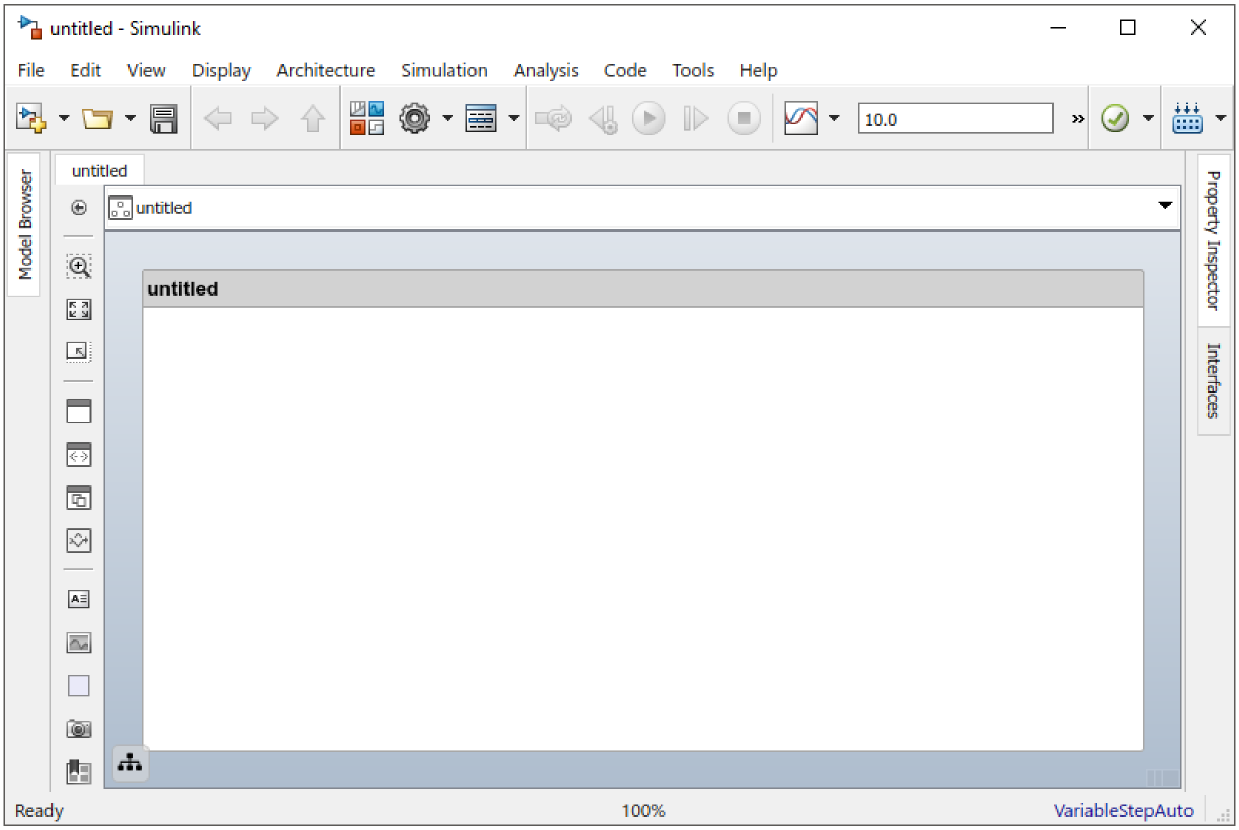


Create Architecture Model

Type `systemcomposer` at the MATLAB®command line to start System Composer. This opens the Simulink start page:

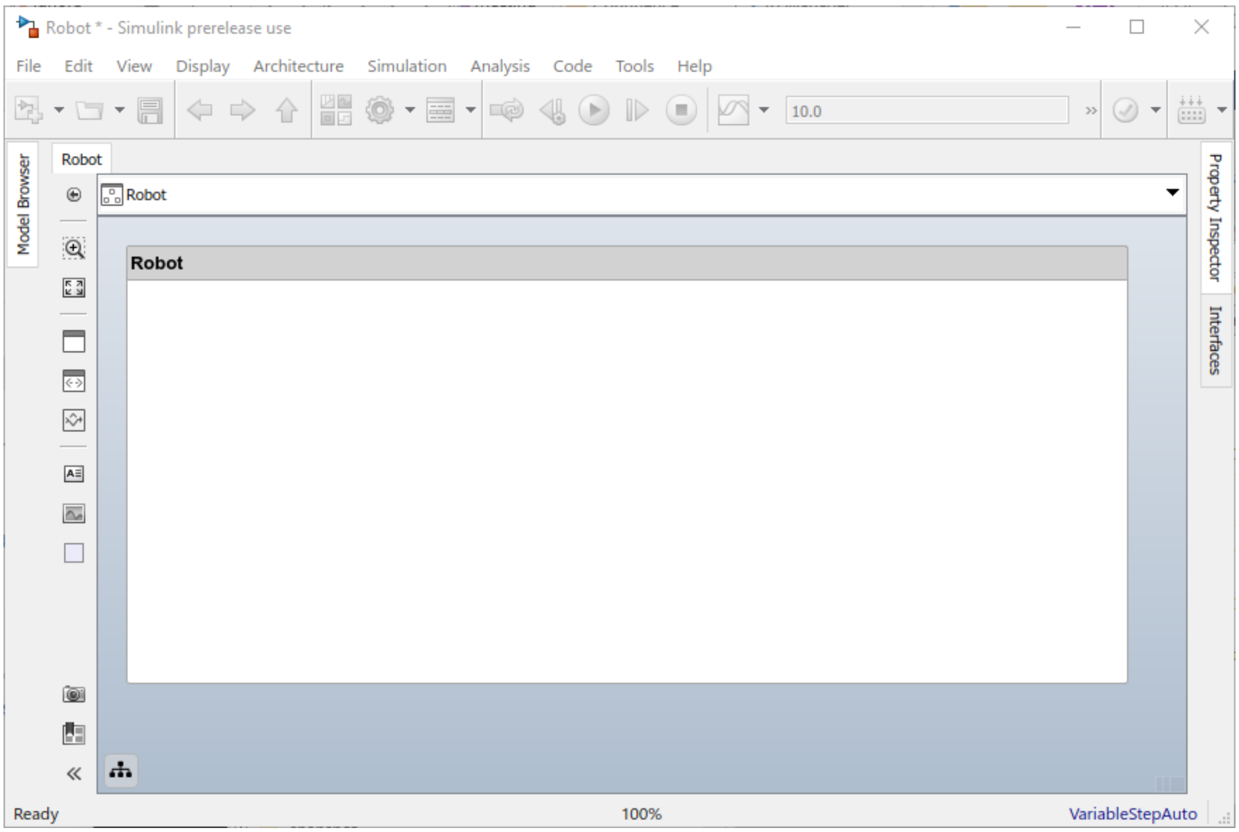


Start composing by clicking **Architecture Model**.



You can identify an architecture model by the badge on the lower left corner, the component palette on the left side, and the **Architecture** menu.

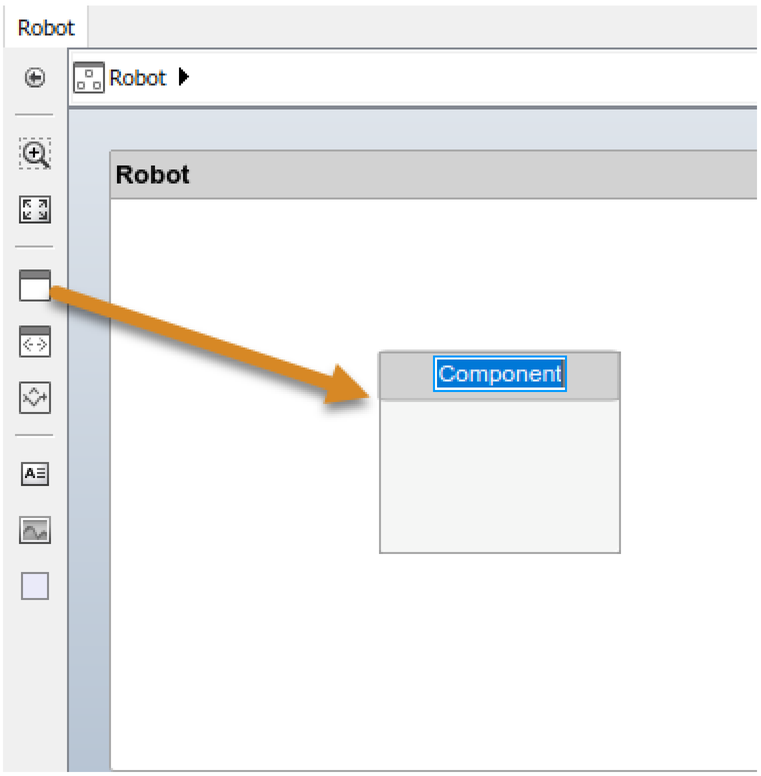
The box in the composition window represents the mobile robot architecture. Double-click the title of the architecture and name it `Robot`:



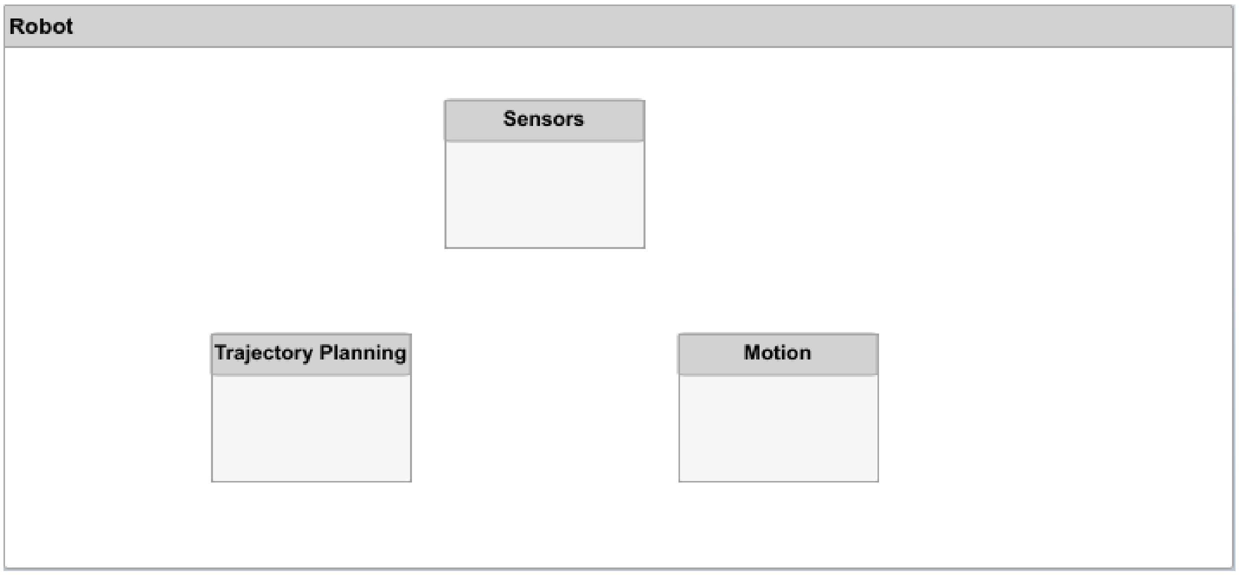
The name of the model reflects the name of the architecture. Save the model using **File > Save**.

Draw Components

Add a component by clicking and dragging a Component from the palette.

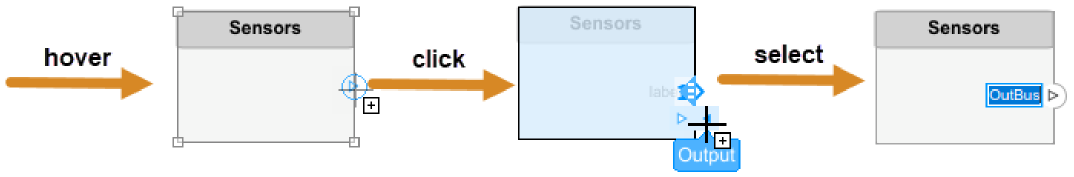


Type the name of the component, `Sensors`, in the text box that becomes active. Also add a `Motion` and a `Controller` component. Component names must be proper variable names. Click components and drag them to move to complete the layout.

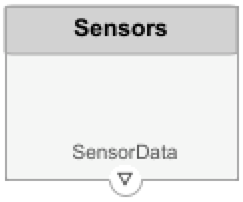


Create Ports and Connections

Create ports on components by clicking one of the edges. Release the mouse button to see direction options, and select a direction to add the port. Name the port. You can add ports on any edge, in any direction.

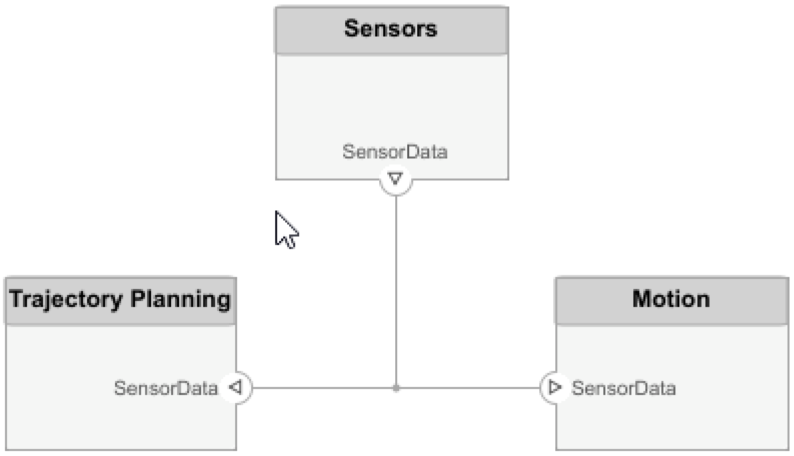


Create an output port at the right hand side of the Sensors component called `SensorData`. Port names must be proper variable names. Select the port and move it to the bottom edge using the down arrow.

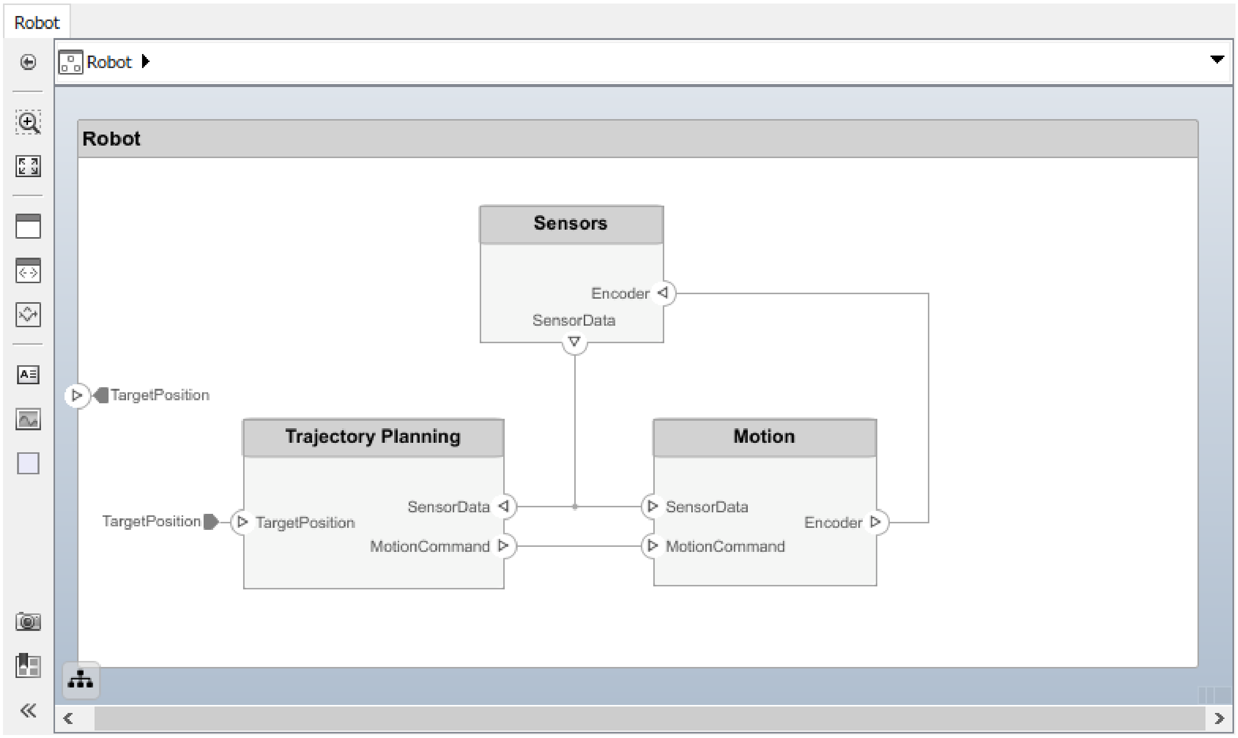


Connect this port to the left side of the Motion component by dragging a line from the port. Release the mouse button when you see an input port created at the destination block. By default, this new port has the same name as the source port.

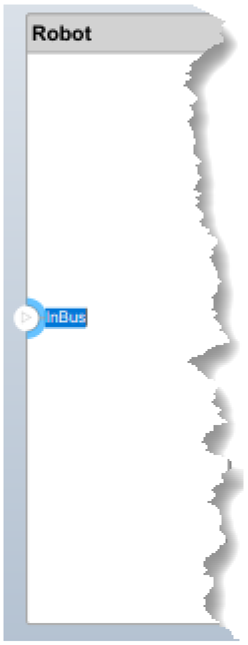
Branch the connection by right-clicking and dragging it to the `Trajectory Planning` component.



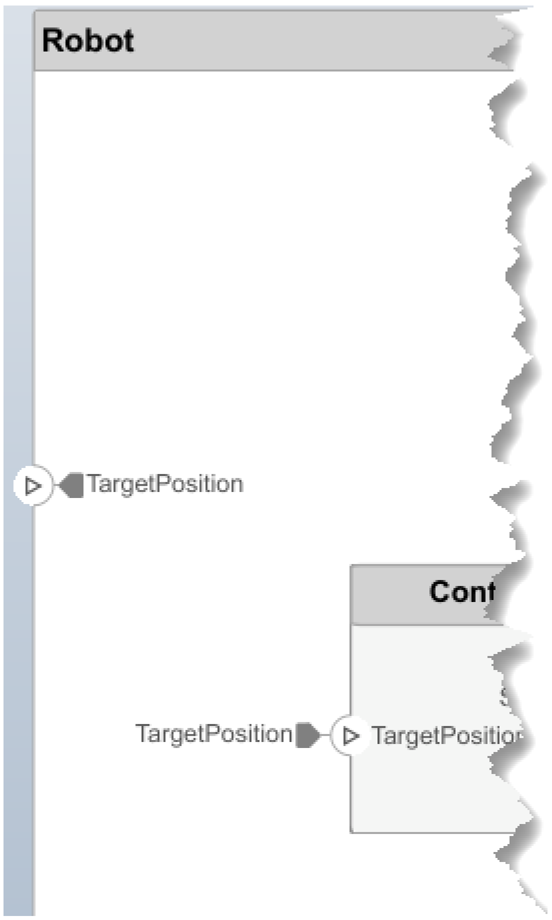
Complete connections as given in the sketch.



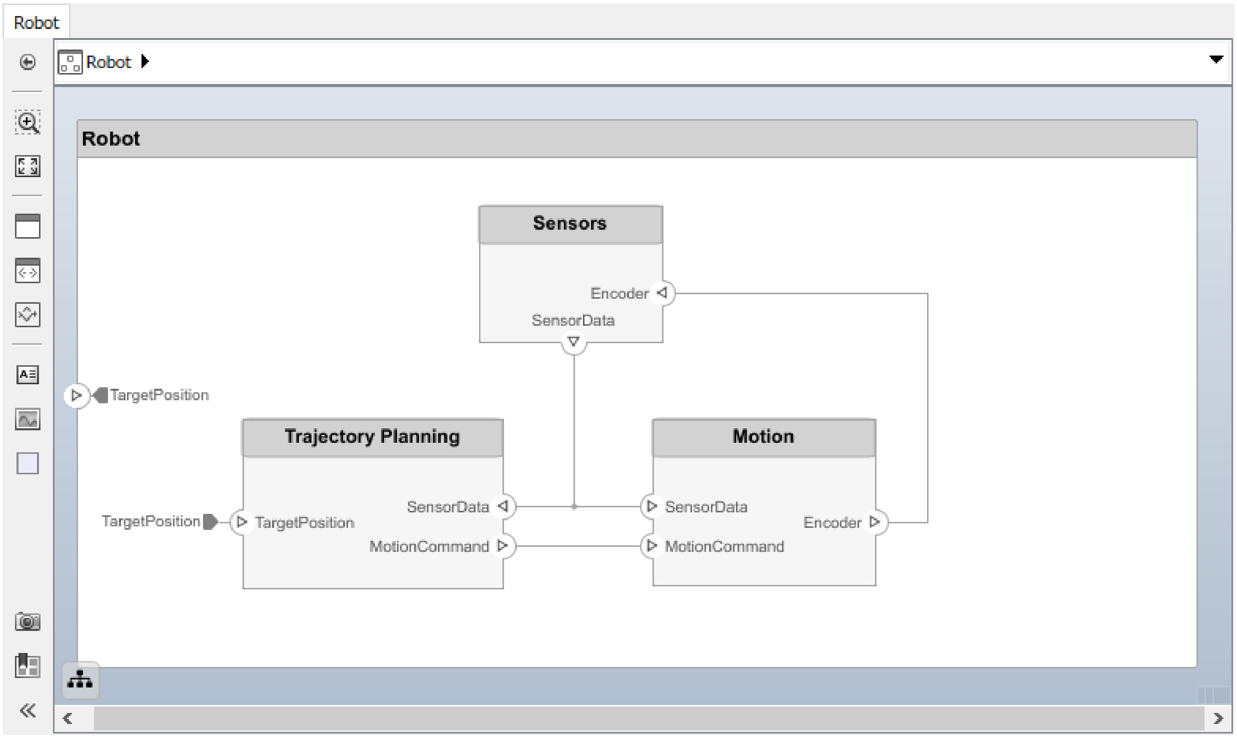
The architecture itself can also have ports. In this example, the target position for the robot is provided by a computer, external to the robot architecture. Represent this with a port on the architecture. Click the left edge of the architecture and type the port name `TargetPosition`.



You can connect an architecture port to a component by dragging a line. Connections to or from an architecture port appear as tags.



Complete the connections between components as given in the sketch:



Edit Interfaces

Specify the data flow between system components by structuring the data interface with data types, units, dimensions, etc. An interface can be as simple as sending an integer value, but it can be a set of numbers, an enumeration, a combination of numbers and strings, or a bundle of other, predefined interfaces.

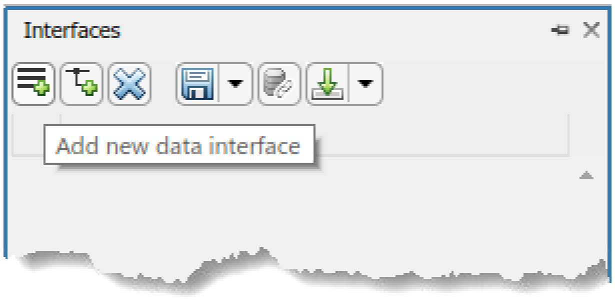
Consider the interface between the Sensor component and the Controllercomponent. The sensor data consists of the following:

- Position data from two motors
- Obstacle proximity data from two sensors
- A time stamp it adds to this data set

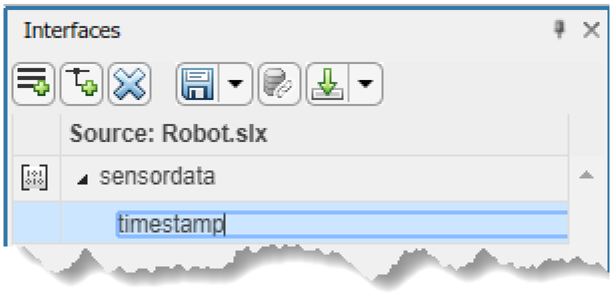
The data has these specifications:

Name	Data Type	Unit
Time stamp	double	seconds
Motor 1 Position	double	degrees
Motor 2 Position	double	degrees
Sensor1 proximity distance	double	meters
Sensor1 proximity direction	double	degrees
Sensor2 proximity distance	double	meters
Sensor2 proximity direction	double	degrees

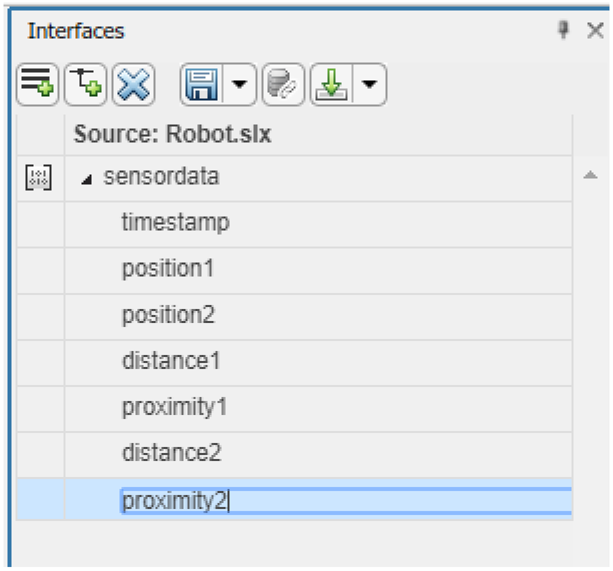
Select **Architecture > Interfaces** tab to start defining an interface. Click the button.



Type in the name of the interface, `sensordata`. Add an element to the selected interface: Click and type `timestamp`.



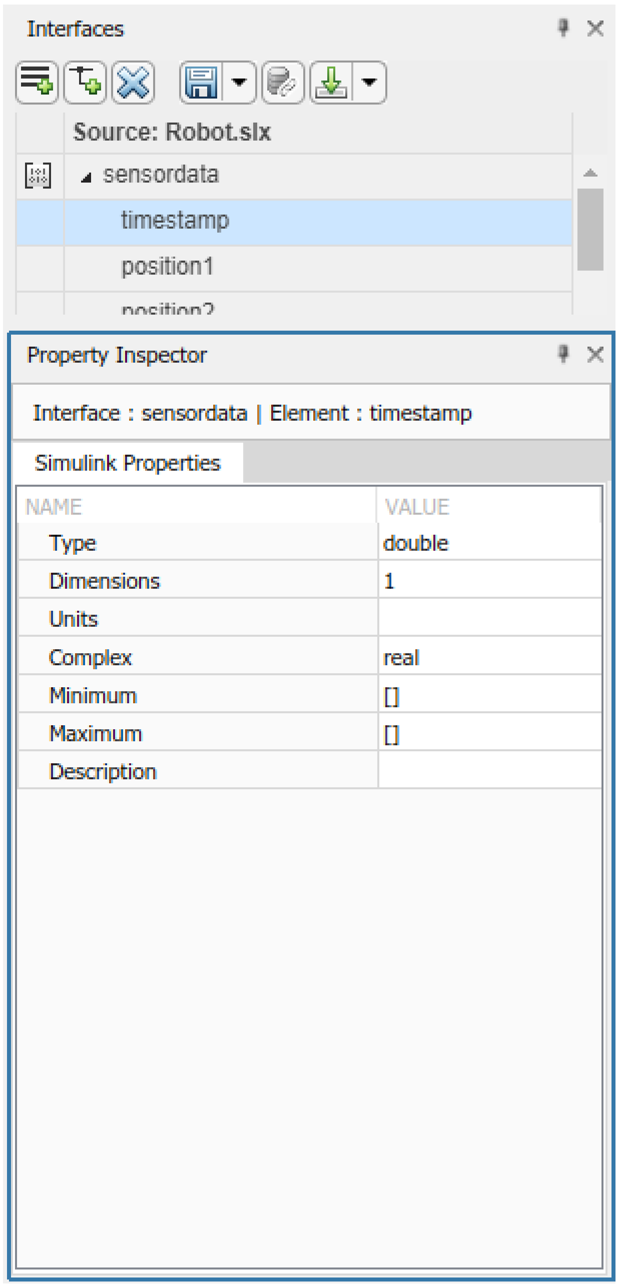
Keep adding elements to the interface by clicking the button multiple times. Double-click the interface elements to edit names as shown:



Source: Robot.slx

- sensordata
  - timestamp
  - position1
  - position2
  - distance1
  - proximity1
  - distance2
  - proximity2

Edit the properties of an interface element using the **Property Inspector**. Right click an element and select **Inspect Properties**.



Source: Robot.slx

sensordata
timestamp
position1
position2

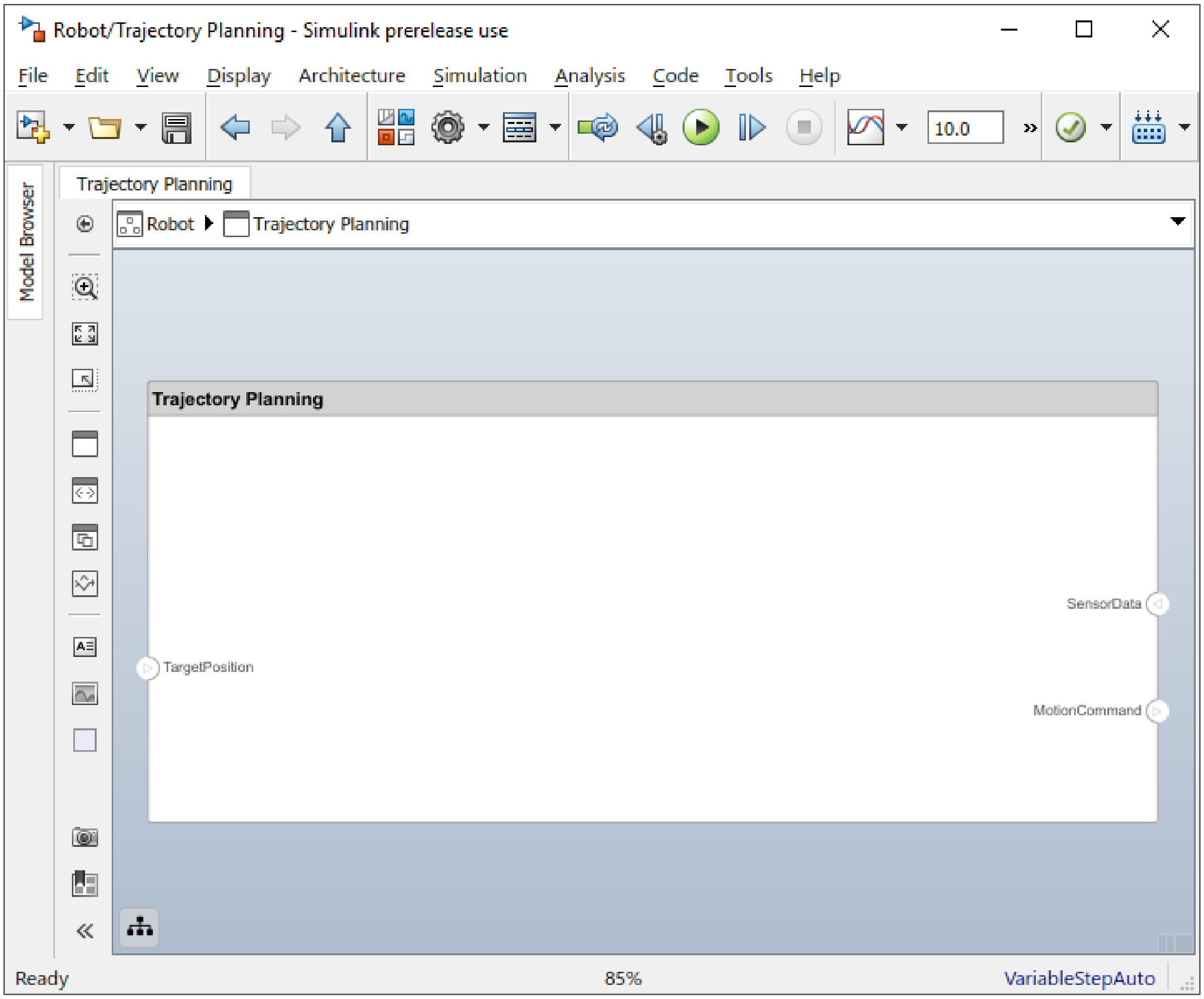
Interface : sensordata | Element : timestamp

NAME	VALUE
Type	double
Dimensions	1
Units	
Complex	real
Minimum	[]
Maximum	[]
Description	

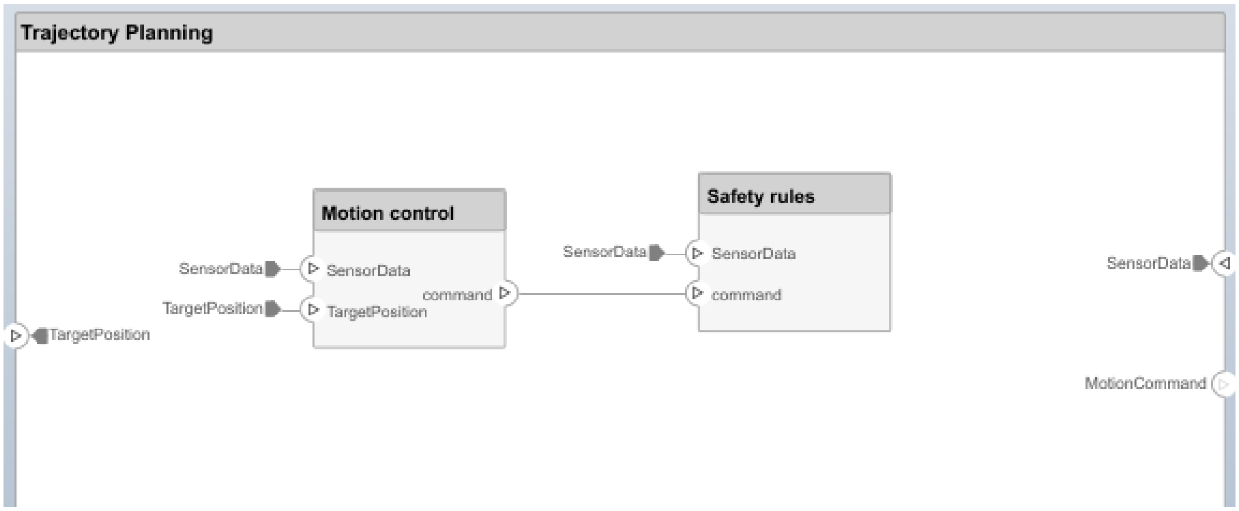
For example, add units for each element as given in the specification.

Decompose Components

Each component can have an architecture of its own. Start decomposing a component into its subcomponents by double-clicking. For example, double-click Trajectory Planning to define its architecture. You can see where the component lies in the hierarchy in the title or the **Model Browser**.

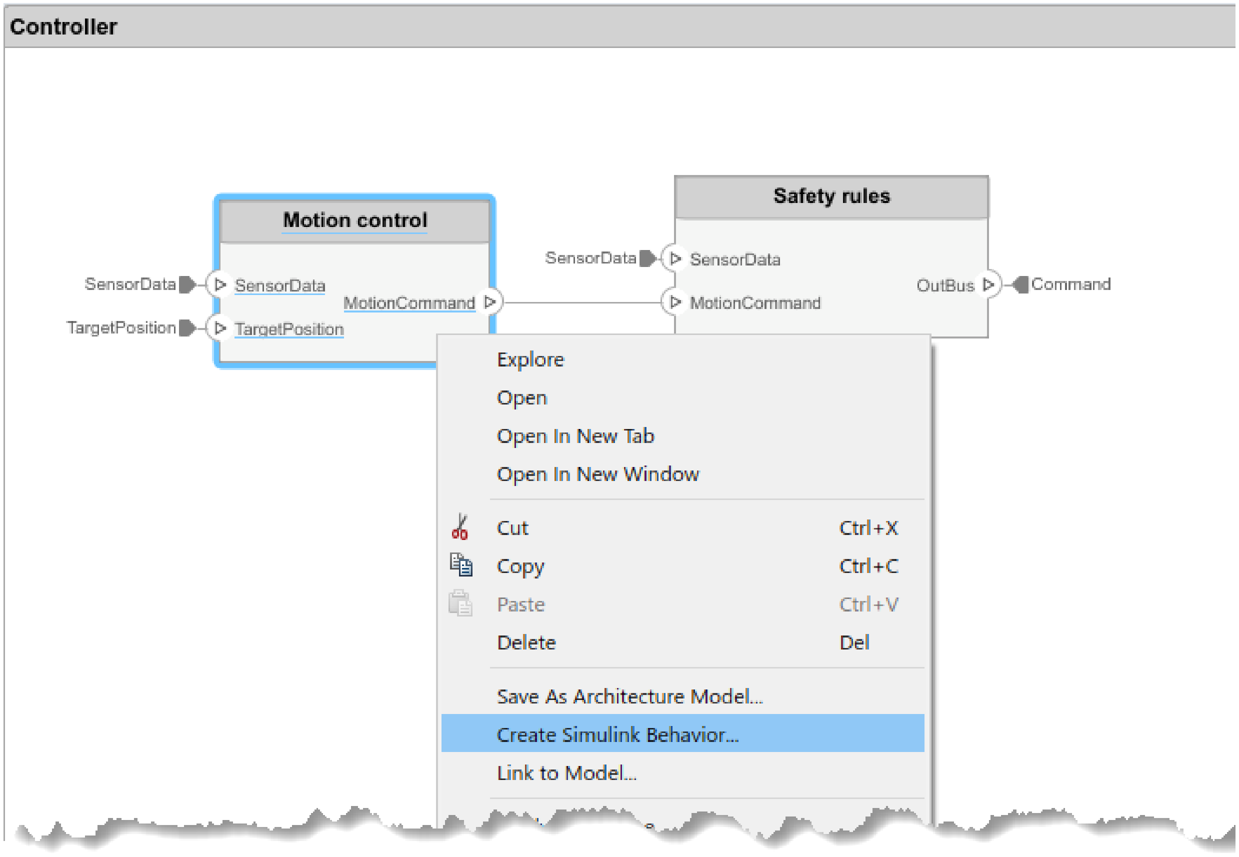


This component first computes an ideal position and velocity command using the motor position data that is part of the `SensorData` interface, and then uses the obstacle proximity information from the same interface to condition this motion command according to some safety rules. Add Motion control and Safety rules components as part of the Trajectory Planning architecture and connect them as shown:

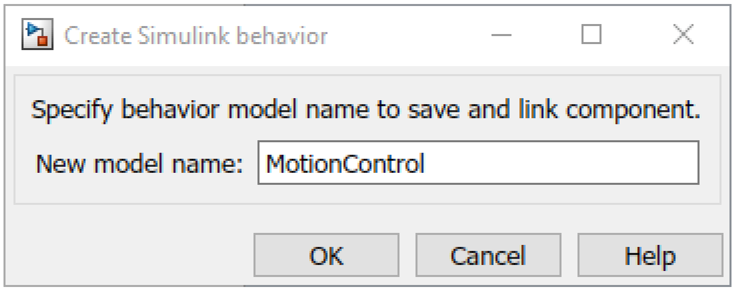


Implement Component Behavior

If you have a component that represents a single functional unit that does not need further architectural decomposition, you can define its behavior in Simulink. Right-click the component and select **Create Simulink Behavior**.



Type the name for a new Simulink model and click **OK**.

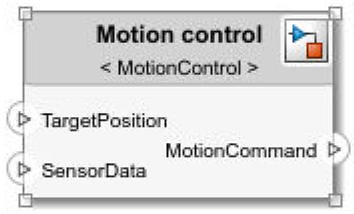


Specify behavior model name to save and link component.

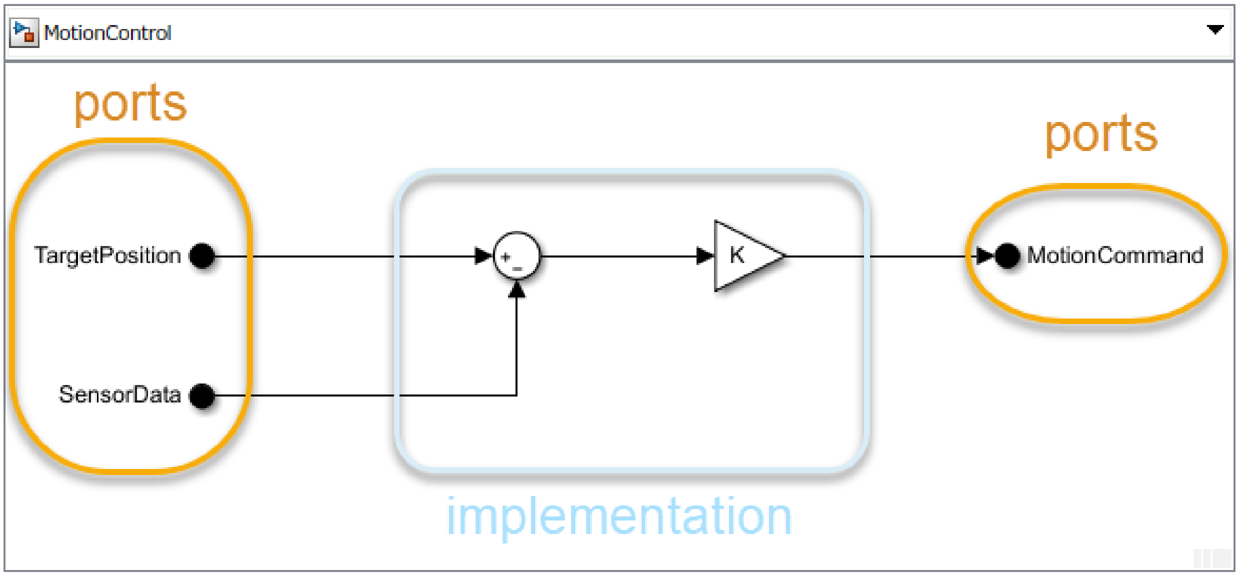
New model name: MotionControl

This creates a new model in the current directory, and adds the Simulink icon and reference model name on the component to indicate the link. The generated behavior model reflects the ports from the component in the architecture model and allows you to define the behavior in Simulink.

Architecture model:

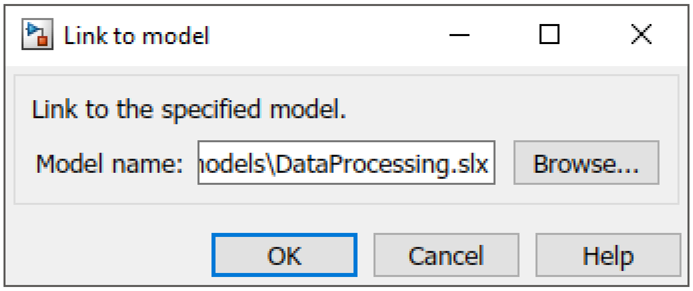


Behavior model:



Link to an Existing Simulink Behavior Model

You can link to an existing Simulink behavior model from a System Composer component, provided that the component is not already linked to a reference architecture. Right-click the component and select **Link to Model**. Type in or browse for the name of a Simulink model.



Any subcomponents and ports that are present in the components get deleted when the component links to a Simulink model.

See Also

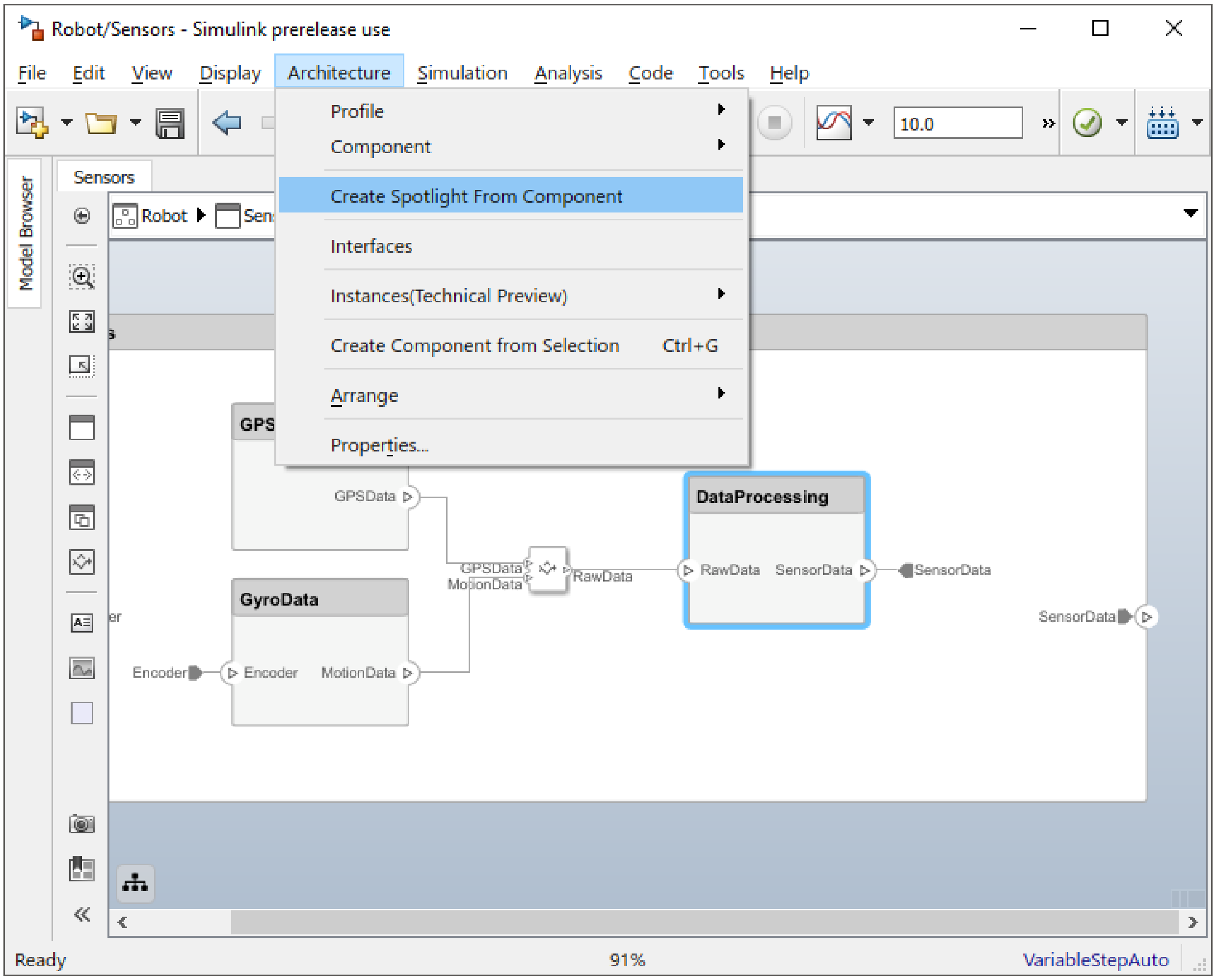
More About

- “Analyze Architecture Model” on page 1-30
- “Create Spotlight Views”

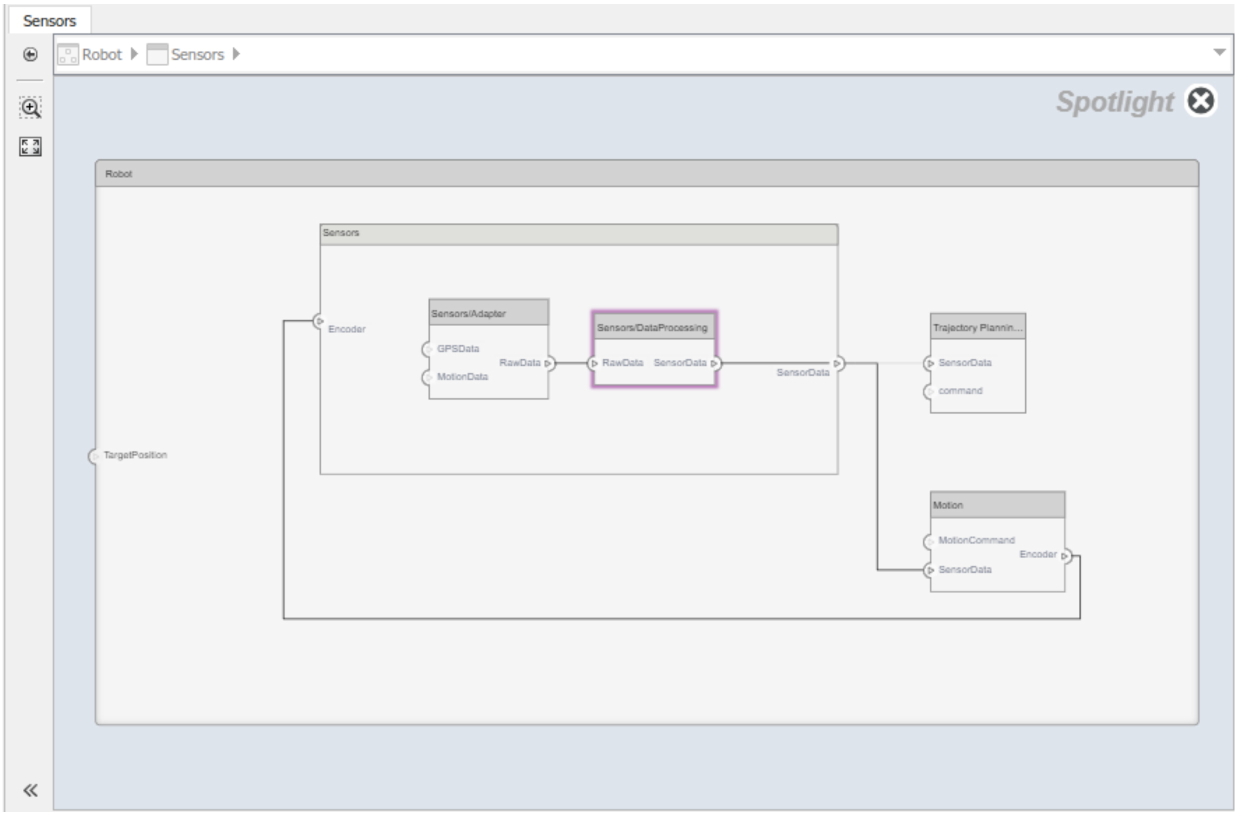
Inspect Components in Custom Views

View the hierarchy and connectivity of a component in a specialized view.

Open the architecture model. Double-click the Sensors block and select the DataProcessing component. Select **Architecture > Create Spotlight from Component**.

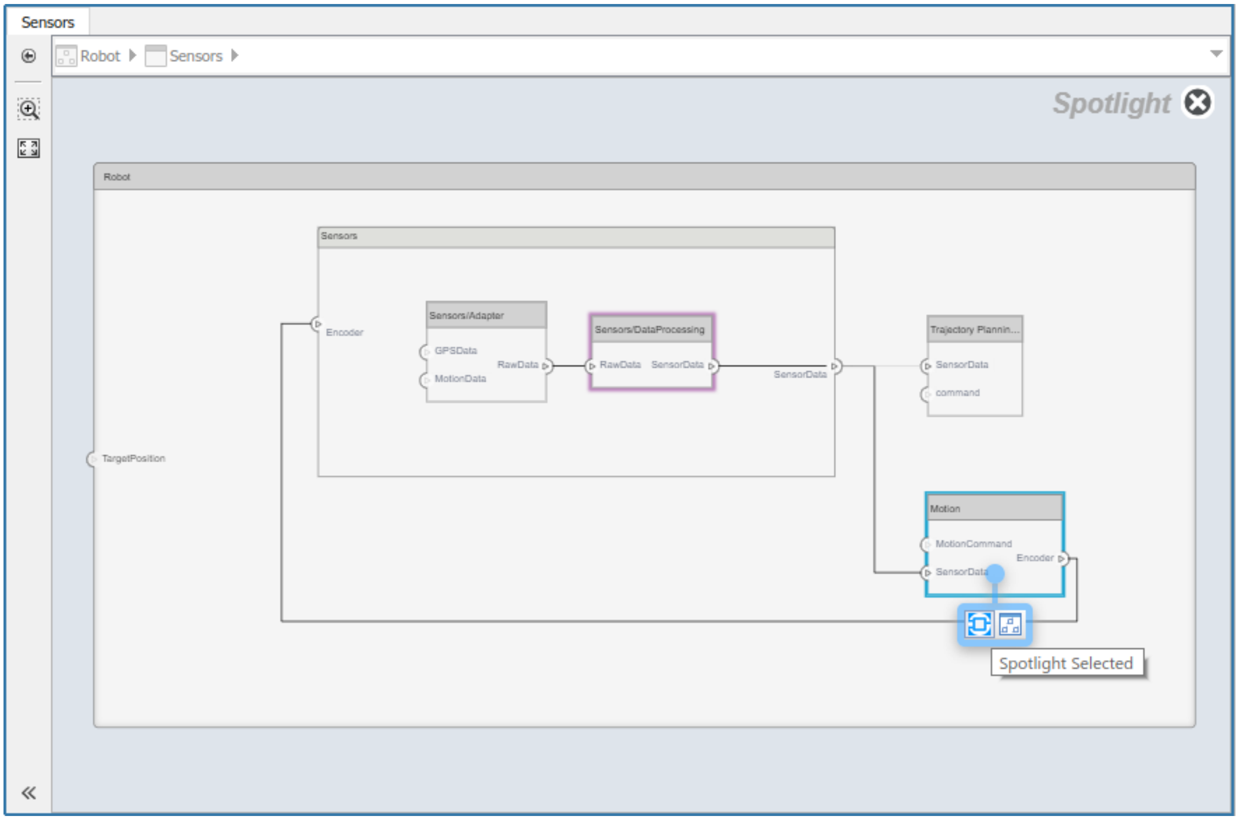


The spotlight view launches and shows all model elements to which the DataProcessingcomponent connects. The spotlight diagram is laid out automatically and cannot be edited.

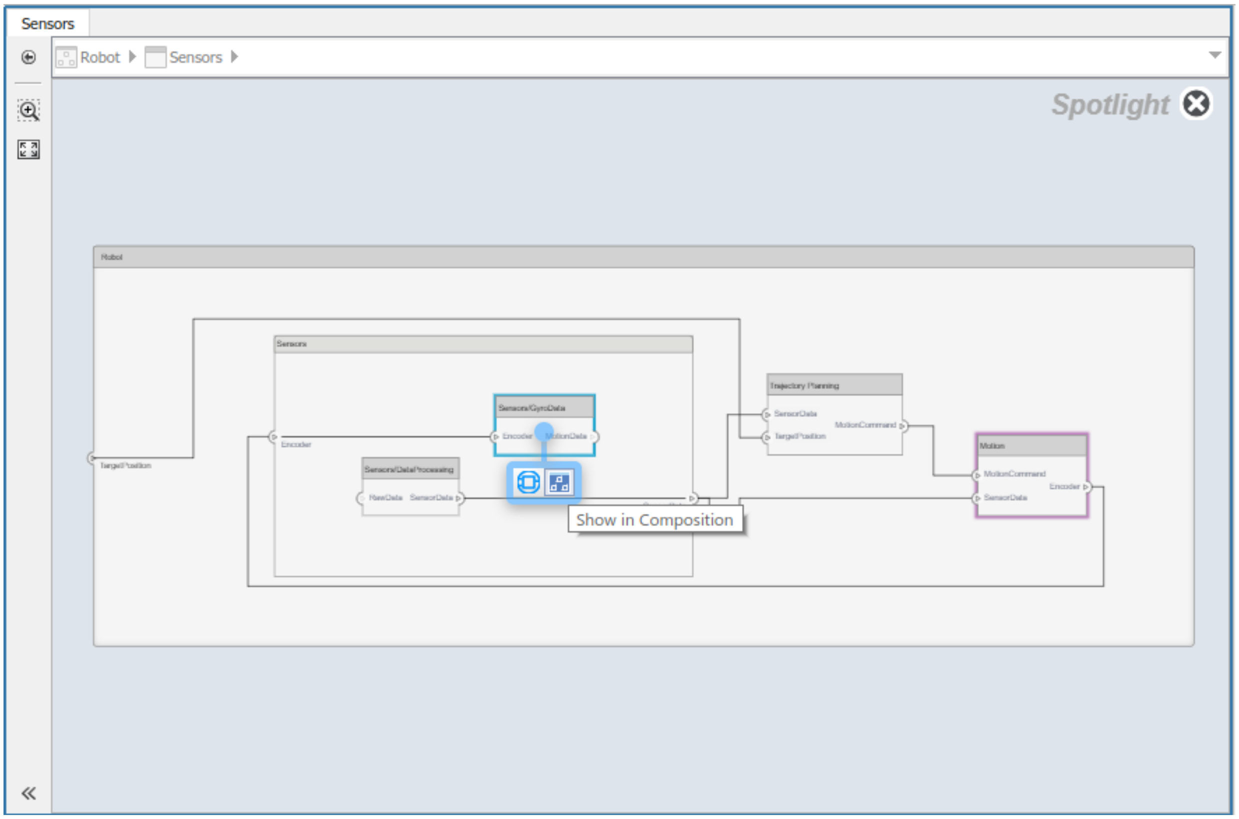


Spotlight views are transient, they are not saved with the model.

While in the spotlight view, put the component in the spotlight. Select the Motioncomponent and click .



Return to the architecture model view by clicking the ⊗ icon. To view the architecture at the level of a particular component, select the component and click the ⊞ icon.



See Also

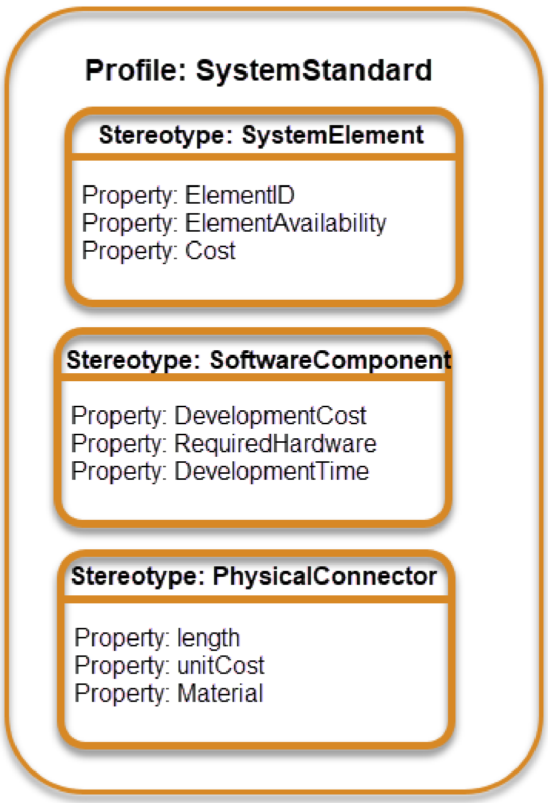
More About

- "Analyze Architecture Model" on page 1-30

Analyze Architecture Model

In this section...

A profile contains a set of model element stereotypes with custom properties. A stereotype can be applicable to components, ports, connections, and architectures; or it can be applicable to a specific element type, such as components. When a model element has a stereotype, you can specify property values as part of its architectural definition. Stereotypes and properties also make analysis of an architecture possible.


Profile: SystemStandard
Stereotype: SystemElement
Property: ElementID
Property: ElementAvailability
Property: Cost
Stereotype: SoftwareComponent
Property: DevelopmentCost
Property: RequiredHardware
Property: DevelopmentTime
Stereotype: PhysicalConnector
Property: length
Property: unitCost
Property: Material

Each profile comes with a set of stereotypes, and each stereotype comes with a set of properties.

The goal of this example is to compute the total cost of the system given the cost of its parts, and the example profile is limited to this goal.

Load Robot System Profile

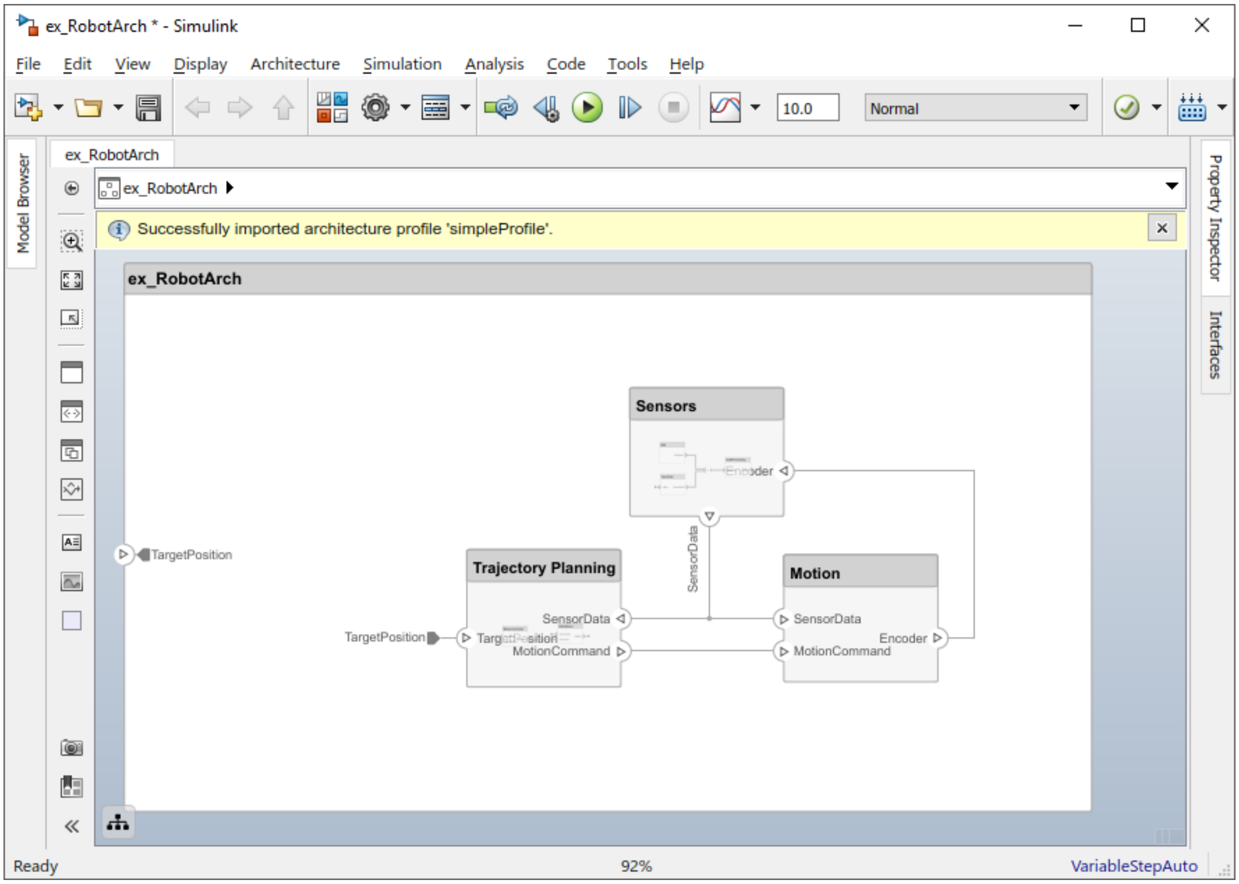
Load a profile to make stereotypes available for model elements. On the model, select **Architecture > Profile > Import Profile** and browse to the profile in `<matlabroot>\toolbox\systemcomposer\examples`.

```
simpleProfile.xml
```

This profile contains these stereotypes:

Stereotype	Application	Properties
sysGeneral	components, ports, connectors	`ID` (integer, no units)
		`Note` (string, no units)
sysComponent	components	`weight` (double, kg)
		`unitPrice` (double, USD)
sysConnector	connectors	`length` (double, m)
		`weight` (double, kg/m)
		`unitPrice` (double, USD/m)

Importing the profile makes stereotypes available to their applicable elements.


ex_RobotArch * - Simulink
File Edit View Display Architecture Simulation Analysis Code Tools Help
10.0
Normal
ex_RobotArch
Model Browser
Property Inspector
Interfaces

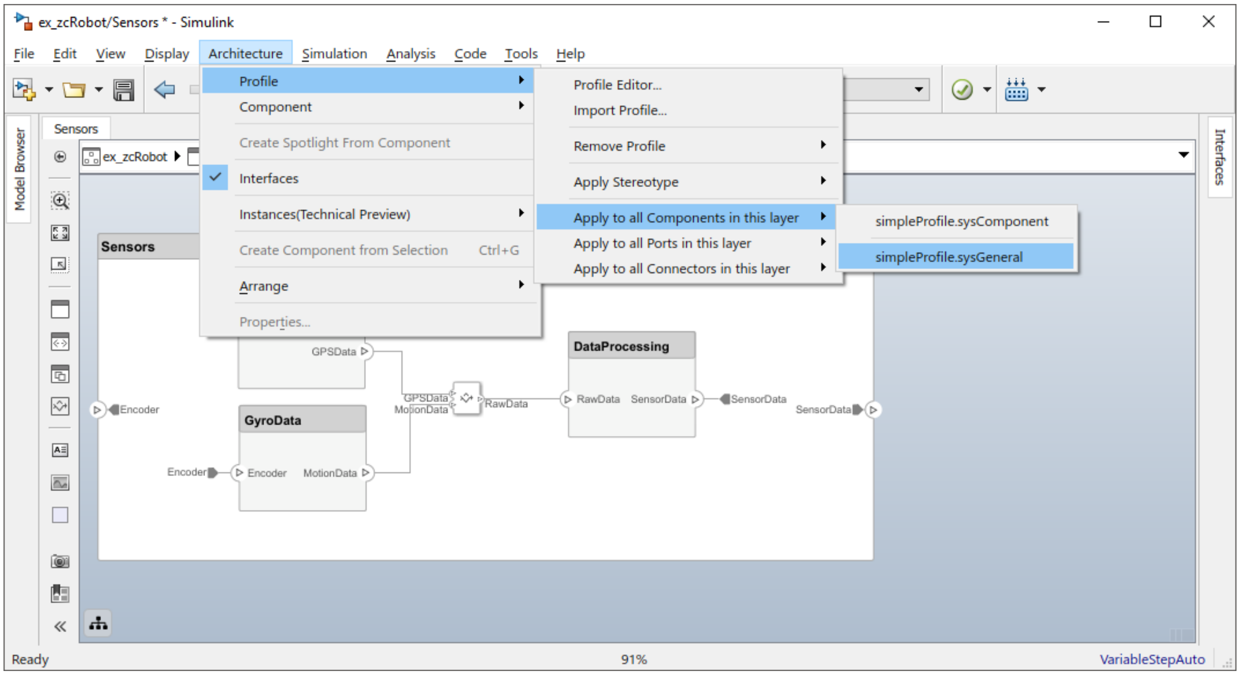
Successfully imported architecture profile 'simpleProfile'.

`sysGeneral` is a general stereotype, applicable to all element types, that enables adding generic properties such as an `ID` that helps identify the element throughout the design and implementation process, and a `Note` that project members can use to track any issues with the element. `sysComponent` applies only to components, and includes properties such as its weight and cost that contribute to the total weight and cost specifications of the robot system. Similarly, the `sysConnector` stereotype applies to connectors, and includes price and weight properties defined per meter of length. These properties help compute the total weight and cost of the design in this particular example.

Apply Stereotypes to Model Elements

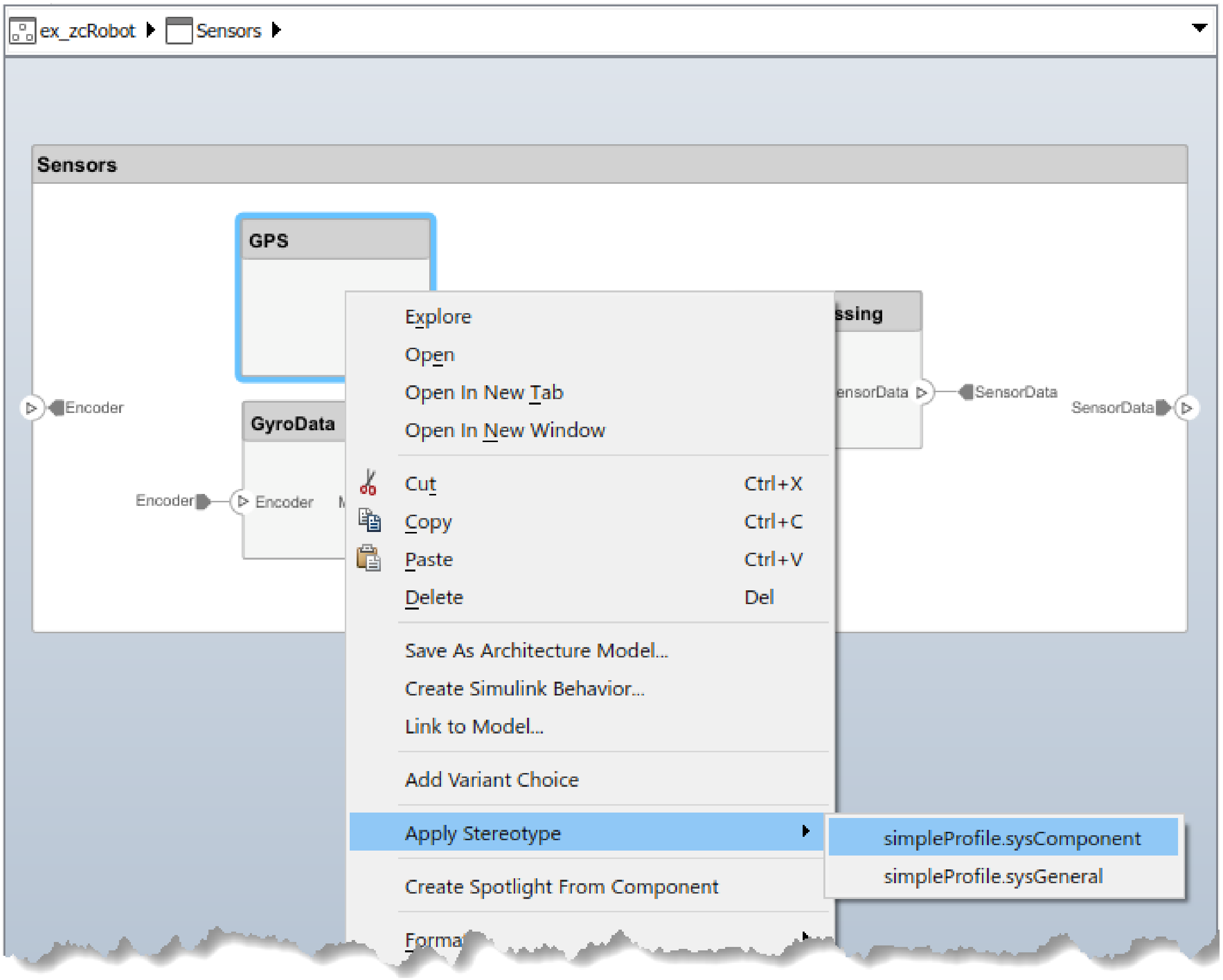
Apply a stereotype to a model element to add custom properties.

1 Open the Sensors component. Select **Architecture > Apply to all Components in this layer > simpleProfile.sysGeneral**. Repeat with **Architecture > Apply to all Connectors in this layer > simpleProfile.sysGeneral**.



2 Select the GPS component of the Sensors component.

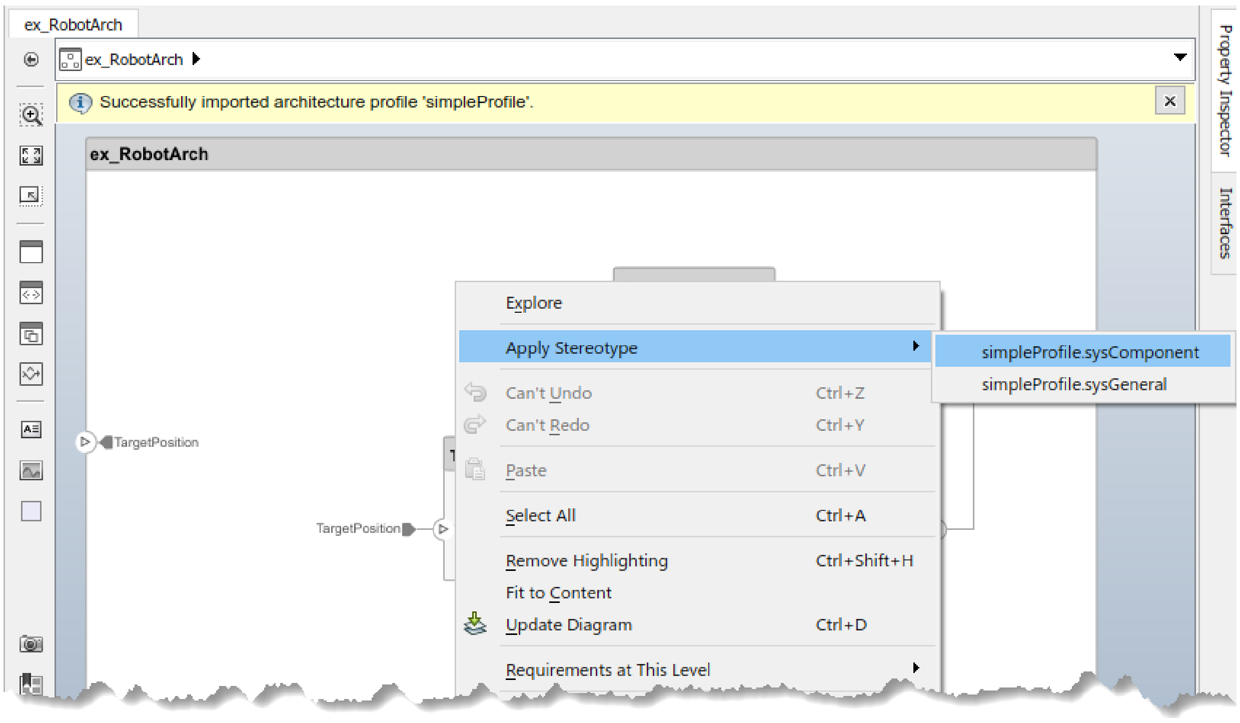
3 Right-click and select **Apply Stereotype > simpleProfile.sysComponent**. Only two of the stereotypes in the profile are available for components. `sysGeneral` is useful for tracking, `sysComponent` is useful for physical properties and cost. You can apply both profiles in this case because `sysGeneral` is applicable to all elements.



4 Apply the `sysComponent` stereotype to all architecture model components in the Sensors level and the Trajectory Planning level.

5 Apply the `sysConnector` stereotype to all connectors in the Sensors layer, the Trajectory Planning layer and the top model layer.

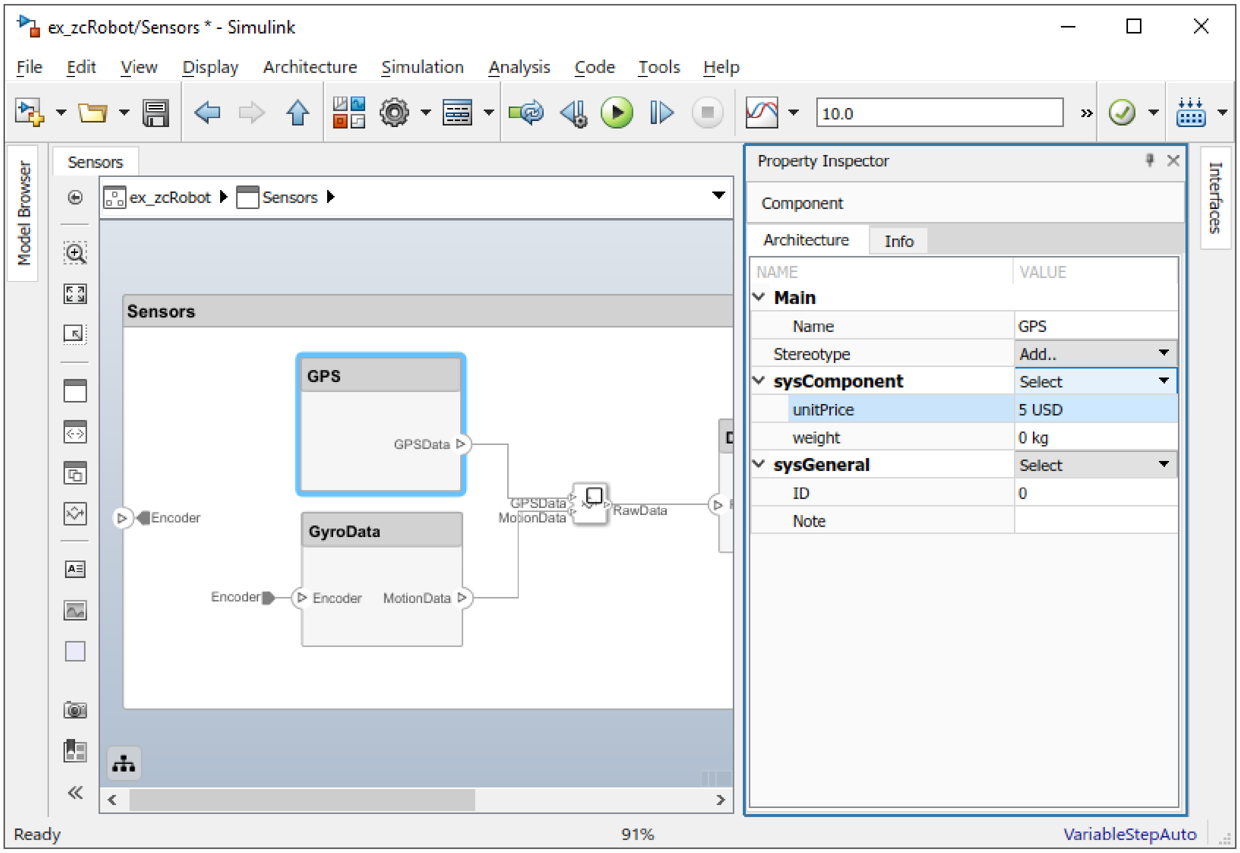
6 Apply the `sysComponent` stereotype to the top level architecture.


Successfully imported architecture profile 'simpleProfile'.

Set Properties

Set the value of properties to enable cost analysis. Follow this example for the GPS module.

1. Select Sensors/GPS.
2. Open the Property Inspector using **View > Property Inspector**.
3. Expand the **sysComponent** stereotype to see the properties.
4. Set **unitPrice** to 5 and press enter.



Property Inspector

Component

Architecture | Info

NAME	VALUE
Main	
Name	GPS
Stereotype	Add..
sysComponent	Select
unitPrice	5 USD
weight	0 kg
sysGeneral	Select
ID	0
Note	

5 Similarly, set the **length** and **unitPrice** properties of the `GPSData` connector.

Complete the model with the values in this table. If a property is not in the table, you can leave it blank as it has no effect on the analysis. Pin the Property Inspector to the editor to make it permanently visible during this operation.

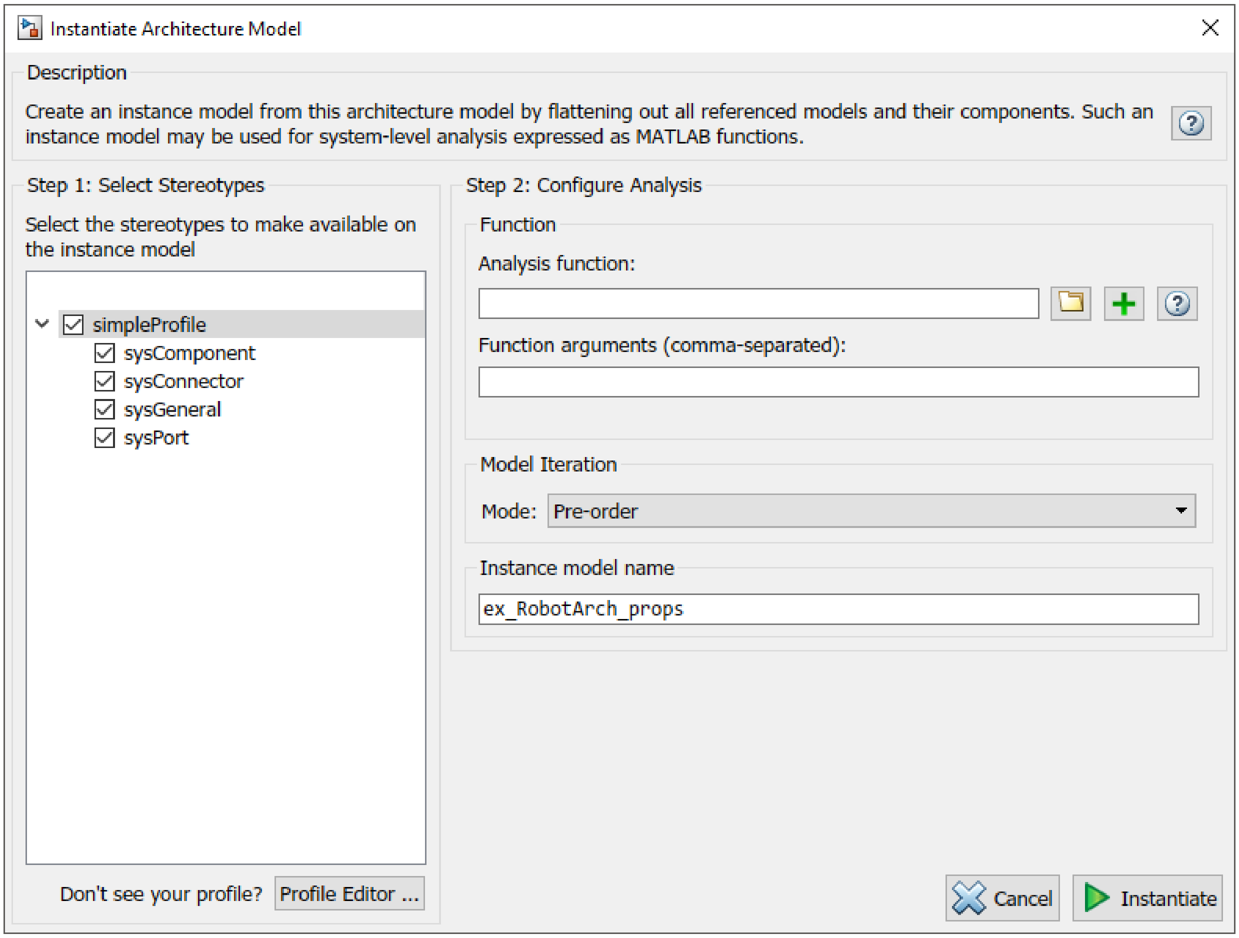
Layer	Element	Property	Value
Top layer	Encoder connector	length	0.5
		unitPrice	0.1
	SensorData connector	length	0.6
		unitPrice	0.2

Layer	Element	Property	Value
	MotionCommand connector	length	0.5
		unitPrice	0.2
	Sensors component	unitPrice	5
	Trajectory Planning component	unitPrice	500
	Motion component	unitPrice	750
Sensors layer	GyroData component	unitPrice	50
	DataProcessing component	unitPrice	500
	GPS component	unitPrice	100
	GPSData connector	length	0.05
		unitPrice	0.1
	MotionData connector	length	0.05
		unitPrice	0.1
	RawData connector	length	0.05
		unitPrice	0.1

Save the model as `ex_RobotArch_props.slx`.

Perform an Analysis

Analyze the total cost for all components in the robot model. Select **Architecture > Analysis > Analyze Architecture Model**. Select the profile for the model.


Instantiate Architecture Model

Description

Create an instance model from this architecture model by flattening out all referenced models and their components. Such an instance model may be used for system-level analysis expressed as MATLAB functions.

Step 1: Select Stereotypes

Select the stereotypes to make available on the instance model

- ☑ simpleProfile
  - ☑ sysComponent
  - ☑ sysConnector
  - ☑ sysGeneral
  - ☑ sysPort

Don't see your profile? Profile Editor ...

Step 2: Configure Analysis

Function

Analysis function:

Function arguments (comma-separated):

Model Iteration

Mode: Pre-order

Instance model name

ex_RobotArch_props

Add an analysis function. Type a new function name without extension, and click ✚.

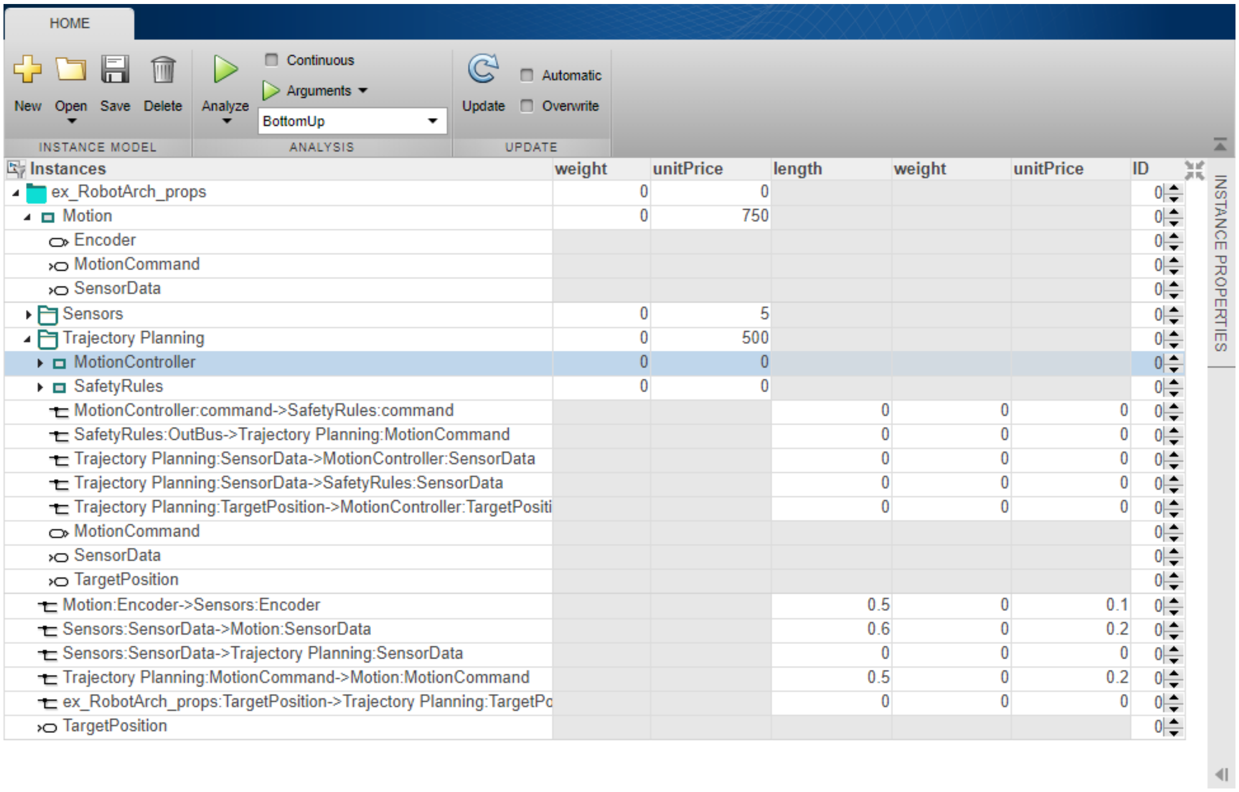
The analysis function includes constructs that get properties from model elements, given as a template. Modify this template to add the cost of individual elements to obtain total cost for their parent architecture. This function computes the cost for one model element:

```
function ex_RobotArch_analysis(instance,varargin)

if instance.isComponent()
    sysComponent_unitPrice = instance.getValue("sysComponent.unitPrice");
```

```
        for child = instance.Components
            comp_price = child.getValue("sysComponent.unitPrice");
            sysComponent_unitPrice = sysComponent_unitPrice + comp_price;
        end
        for child = instance.Connectors
            unitPrice = child.getValue("sysConnector.unitPrice");
            length = child.getValue("sysConnector.length");
            sysComponent_unitPrice = unitPrice*length + sysComponent_unitPrice;
        end
      instance.setValue("sysComponent.unitPrice",sysComponent_unitPrice)
    end
```

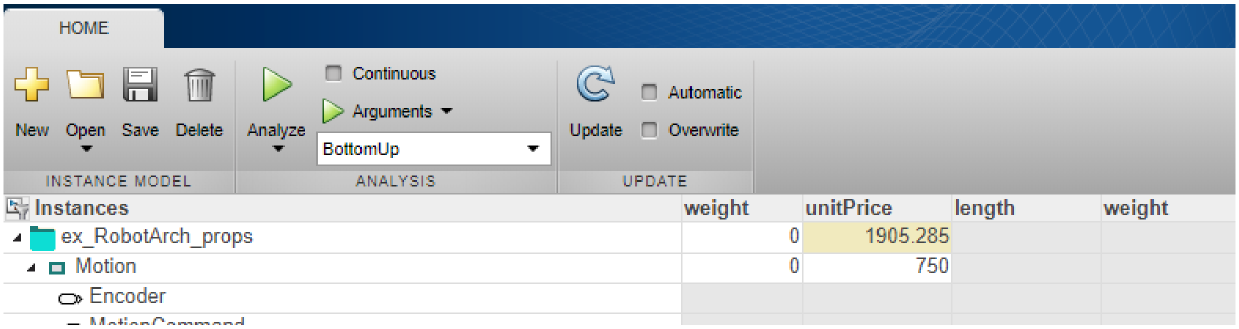
Return to the Analysis screen and click **Instantiate**. The analysis viewer shows the properties of each model element.



Instances	weight	unitPrice	length	weight	unitPrice	ID
ex_RobotArch_props	0	0				0
Motion	0	750				0
Encoder						0
MotionCommand						0
SensorData						0
Sensors	0	5				0
Trajectory Planning	0	500				0
MotionController	0	0				0
SafetyRules	0	0				0
MotionController:command->SafetyRules:command			0	0	0	0
SafetyRules:OutBus->Trajectory Planning:MotionCommand			0	0	0	0
Trajectory Planning:SensorData->MotionController:SensorData			0	0	0	0
Trajectory Planning:SensorData->SafetyRules:SensorData			0	0	0	0
Trajectory Planning:TargetPosition->MotionController:TargetPositi			0	0	0	0
MotionCommand						0
SensorData						0
TargetPosition						0
Motion:Encoder->Sensors:Encoder			0.5	0	0.1	0
Sensors:SensorData->Motion:SensorData			0.6	0	0.2	0
Sensors:SensorData->Trajectory Planning:SensorData			0	0	0	0
Trajectory Planning:MotionCommand->Motion:MotionCommand			0.5	0	0.2	0
ex_RobotArch_props:TargetPosition->Trajectory Planning:TargetPo			0	0	0	0
TargetPosition						0

Select `Bottomup` as the iteration method and click **Analyze**.

The cost of each element is added in a bottom-up manner to find the cost of the system. The result is written to the analysis instance and is visible in the Analysis Viewer.



Instances	weight	unitPrice	length	weight
ex_RobotArch_props	0	1905.285		
Motion	0	750		
Encoder				

See Also

More About

- “Compose an Architecture Model” on page 1-6
- “Define Profiles and Stereotypes”

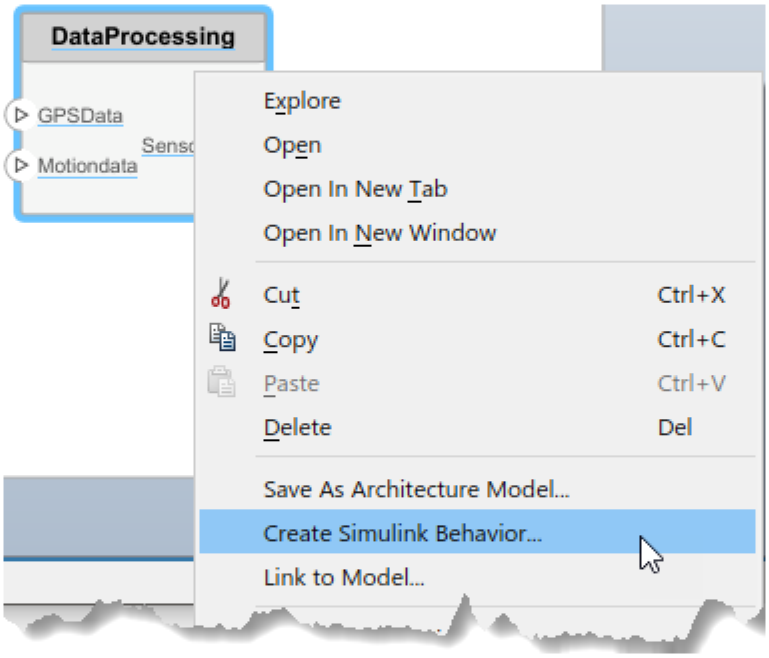
Refactor a Simulink Model as a Composition

Implement Component Behavior in Simulink

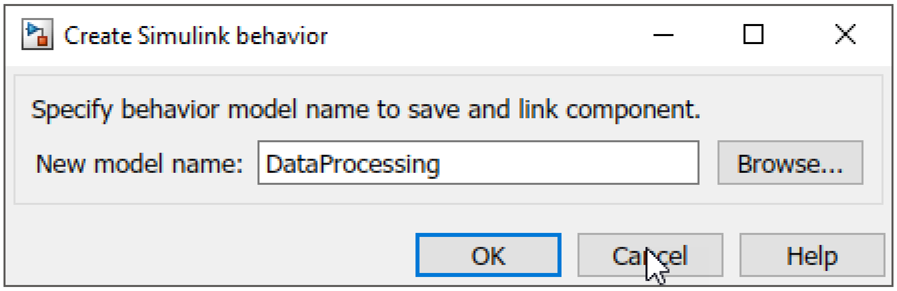
Create and use Simulink models to specify component behavior.

Create a Simulink Behavior Model

When a component does not require further decomposition from an architecture standpoint, you can design its behavior in Simulink. Right-click the component and select **Create Simulink Behavior**.



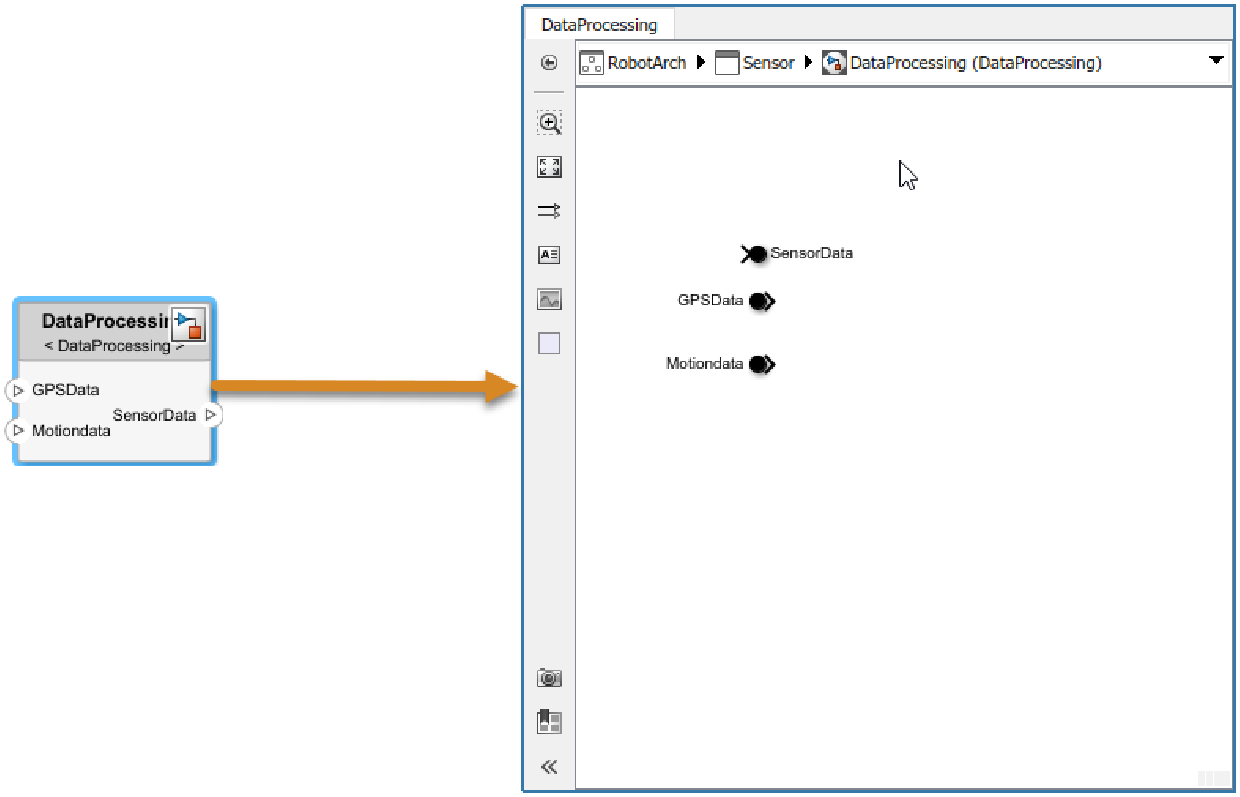
Provide a model name. The default name is the name of the component.


Create Simulink behavior

Specify behavior model name to save and link component.

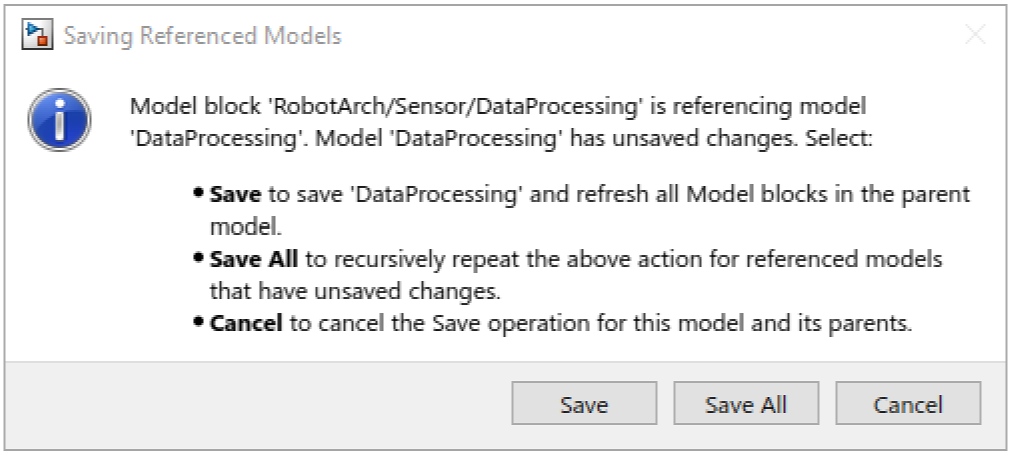
New model name: DataProcessing

- A new Simulink model with the provided name is created. The root level ports of the Simulink model reflect the ports of the component.
- The component in the architecture model is linked to the Simulink model. The Simulink icon on the component indicates this is a Simulink link.



You can now go on to providing specific dynamics and algorithms in the referenced Simulink model. Adding root-level ports in the Simulink model creates additional ports on the referencing component.

You can access and edit a referenced Simulink model by double-clicking the component in the architecture model. When you save the architecture model, all unsavedSimulink behavior models it references must also be saved, and all linked components updated.

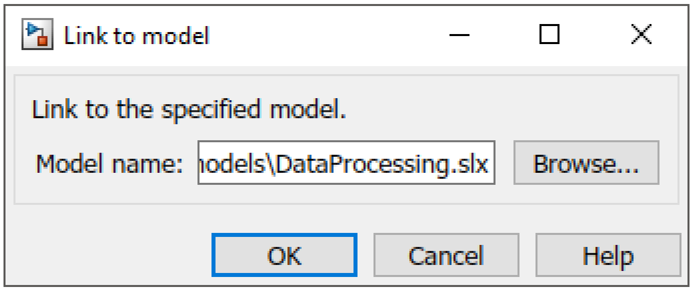


Model block 'RobotArch/Sensor/DataProcessing' is referencing model 'DataProcessing'. Model 'DataProcessing' has unsaved changes. Select:

- **Save** to save 'DataProcessing' and refresh all Model blocks in the parent model.
- **Save All** to recursively repeat the above action for referenced models that have unsaved changes.
- **Cancel** to cancel the Save operation for this model and its parents.

Link to an Existing Simulink Behavior Model

You can link to an existing Simulink behavior model from a System Composer component, provided that the component is not already linked to a reference architecture. Right-click the component and select **Link to Model**. Type in or browse for the name of a Simulink model.



Link to the specified model.

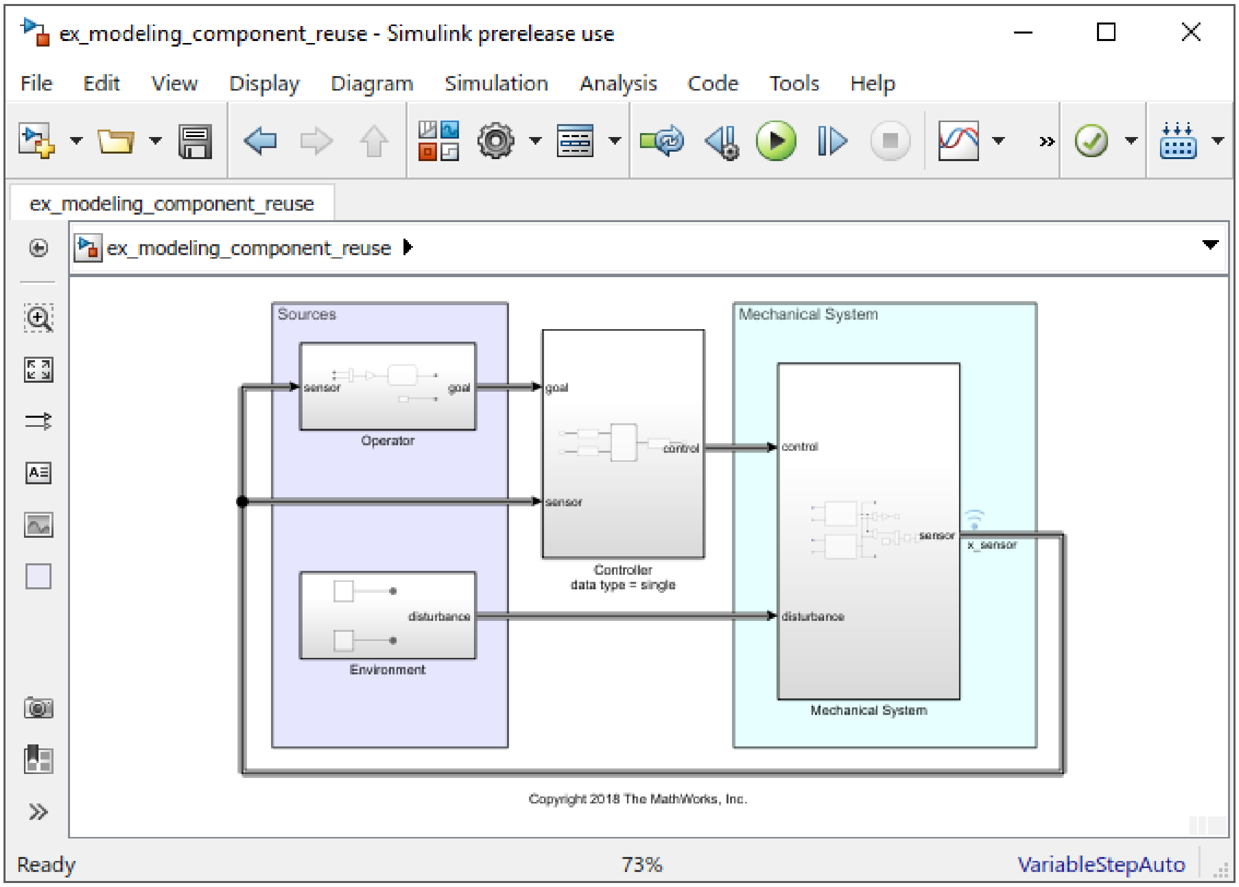
Model name: ıodels\DataProcessing.slx

Any subcomponents and ports that are present in the components get deleted when the component links to a Simulink model.

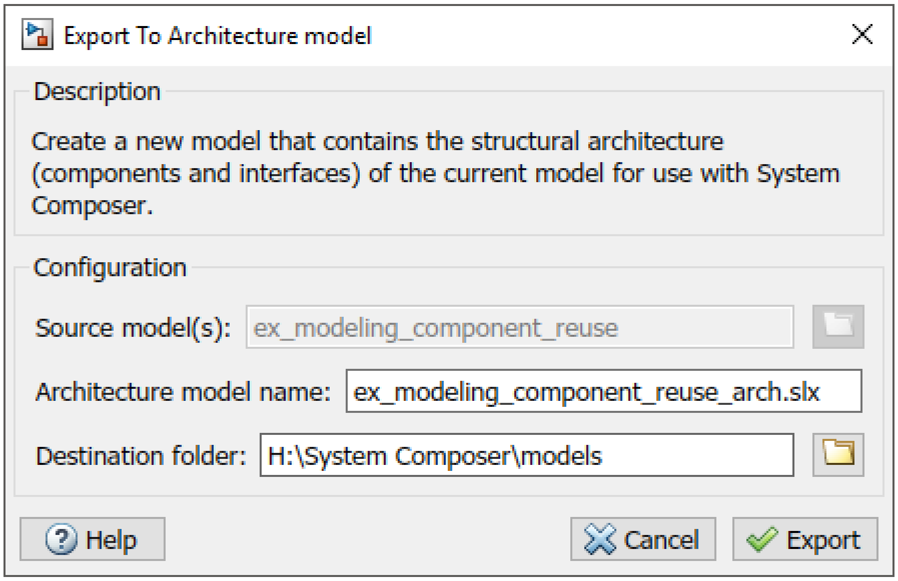
Export Simulink Model as Architecture

You can use System Composer architecture editing and analysis capabilities on Simulink models by extracting the architecture from Simulink models. Model and Subsystem blocks, as well as all ports in a Simulink model represent architectural constructs; whereas all other blocks represent some kind of dynamic or algorithmic behavior. In the resulting architecture model you can choose to represent only architectural constructs, or link to behavior models.

For example, open the model `ex_modeling_component_reuse.slx`.



1. On the Simulink model, select **File > Export Model to > Architecture Model**.
2. Provide a name and path for the architecture model.


Export To Architecture model

Description

Create a new model that contains the structural architecture (components and interfaces) of the current model for use with System Composer.

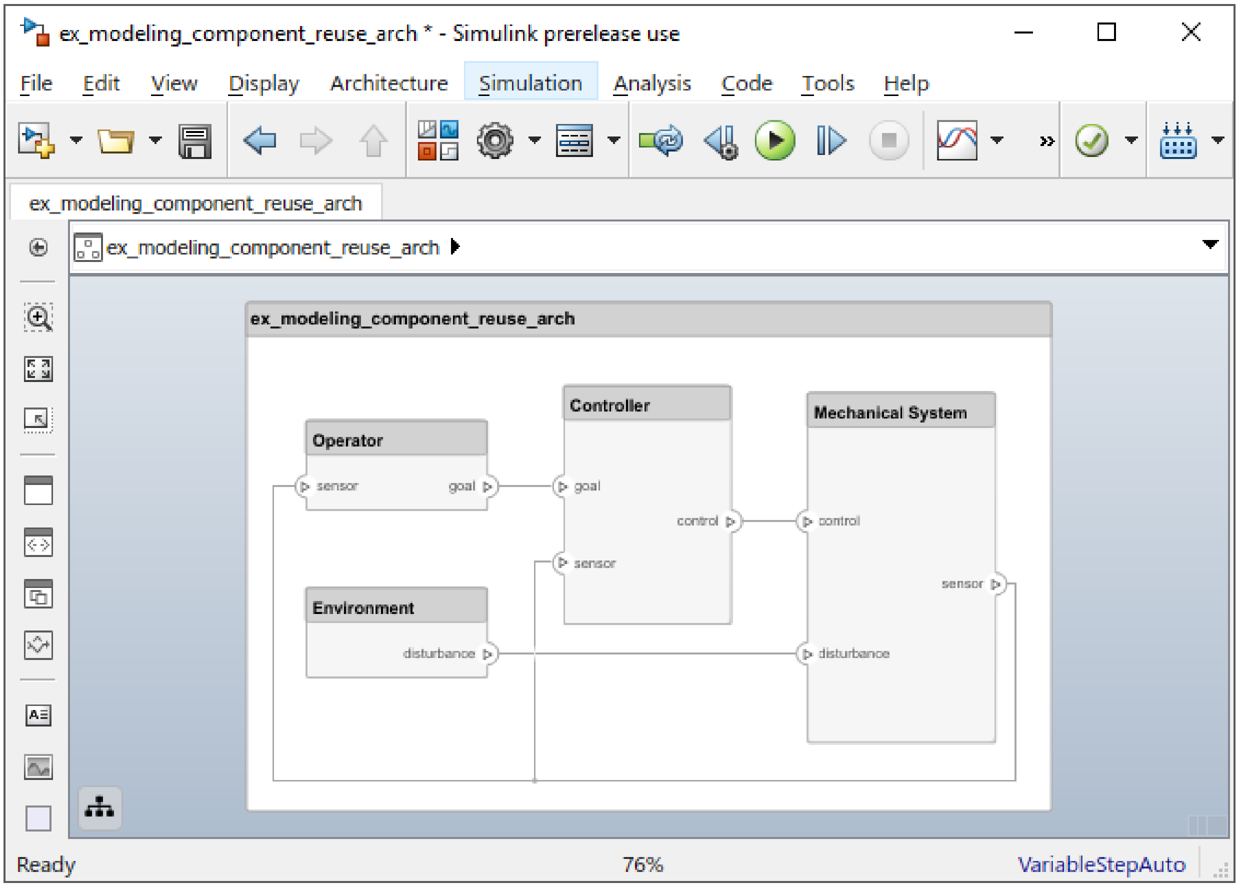
Configuration

Source model(s): ex_modeling_component_reuse

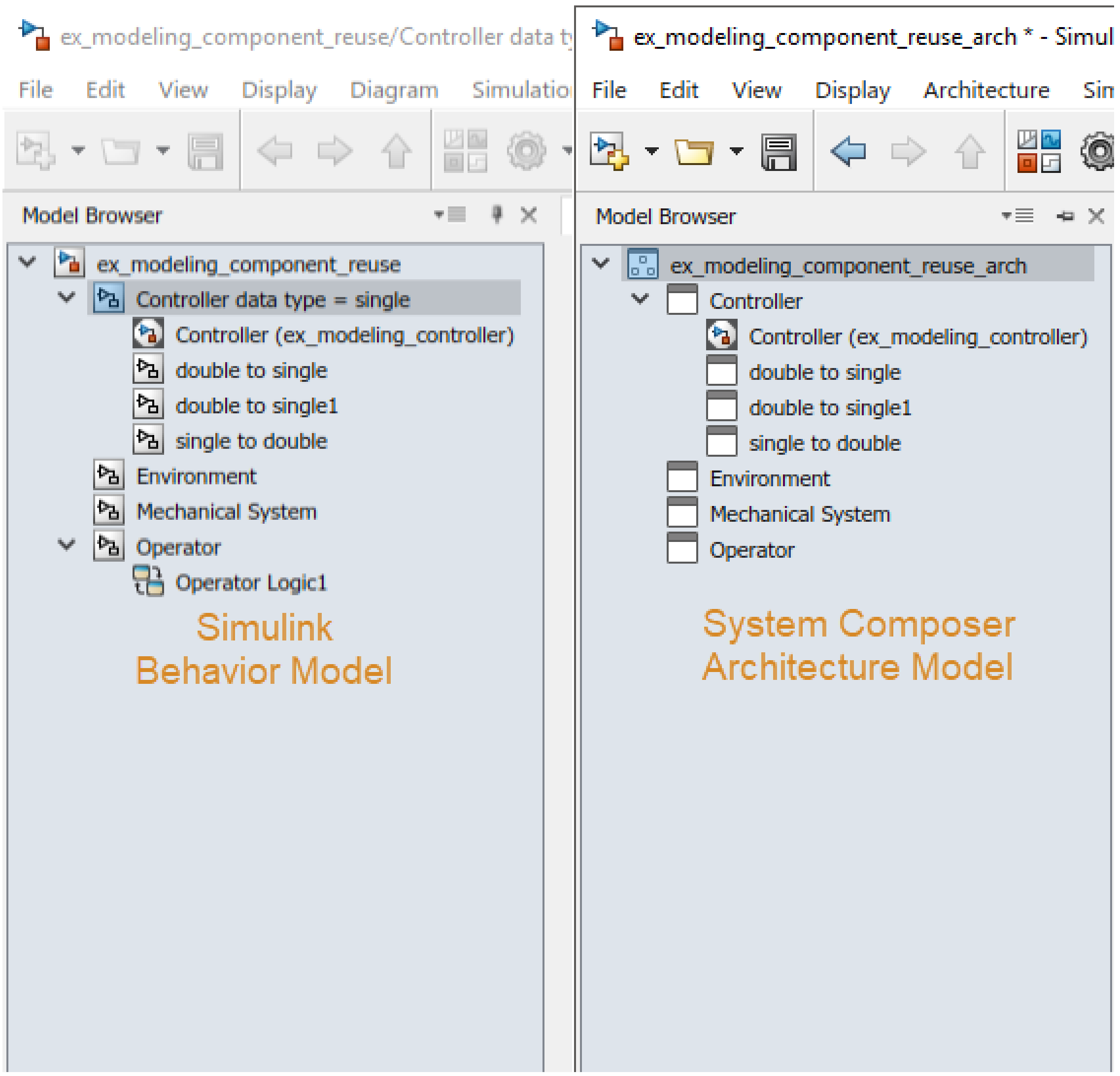
Architecture model name: ex_modeling_component_reuse_arch.slx

Destination folder: H:\System Composer\models

3. Click **Export**.

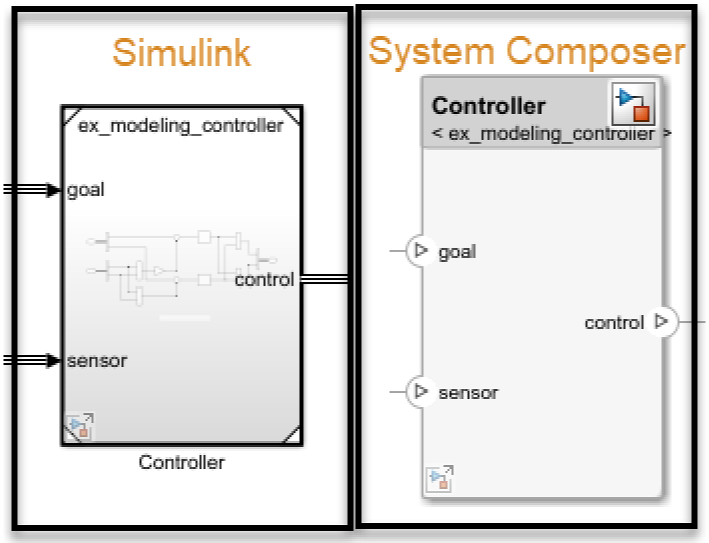

ex_modeling_component_reuse_arch * - Simulink prerelease use
File Edit View Display Architecture Simulation Analysis Code Tools Help
ex_modeling_component_reuse_arch
ex_modeling_component_reuse_arch
ex_modeling_component_reuse_arch
Operator
sensor
goal
Controller
goal
control
sensor
Mechanical System
control
sensor
disturbance
Environment
disturbance
Ready
76%
VariableStepAuto

Each subsystem in the Simulink model corresponds to a component in the architecture model so that the hierarchy in the architecture model reflects the hierarchy of the behavior model.


ex_modeling_component_reuse/Controller data t
File Edit View Display Diagram Simulatio
Model Browser
ex_modeling_component_reuse
Controller data type = single
Controller (ex_modeling_controller)
double to single
double to single1
single to double
Environment
Mechanical System
Operator
Operator Logic1
Simulink Behavior Model
ex_modeling_component_reuse_arch * - Simul
File Edit View Display Architecture Sin
Model Browser
ex_modeling_component_reuse_arch
Controller
Controller (ex_modeling_controller)
double to single
double to single1
single to double
Environment
Mechanical System
Operator
System Composer Architecture Model

The requirements for subsystems and Model blocks in the Simulink model are preserved in the architecture model.

Any Model block in the Simulink model that references another model corresponds to a component that links to that same referenced model.



Buses at subsystem and Model block ports, as well as their dictionary links are preserved in the architecture model.

You can use the exported model to add architecture-related information such as interface definitions, non-functional properties for model elements and analyze the design.

See Also

More About

- "Implement Component Behavior" on page 1-22