

System Composer™

Getting Started Guide



MATLAB® & SIMULINK®

R2020a



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

© COPYRIGHT 2019–2020 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

March 2019	Online only	New for Version 1.0 (Release 2019a)
September 2019	Online only	Revised for Version 1.1 (Release 2019b)
March 2020	Online only	Revised for Version 1.2 (Release 2020a)

1 Compose an Architecture Model

Compose and Analyze a System	1-2
Create an Architecture Model	1-5
Visually Represent the System	1-5
Edit Interfaces	1-12
Decompose Components	1-13
Implement Component Behavior	1-15
Link to an Existing Simulink Behavior Model	1-16
Inspect Components in Custom Views	1-17
Open the Model	1-17
Inspect the Component and Its Connectivity	1-17
Analyze an Architecture Model	1-21
Load Architecture Model Profile	1-22
Apply Stereotypes to Model Elements	1-23
Set Properties	1-25
Perform an Analysis	1-26

2 Refactor a Simulink Model as a Composition

Implement Component Behavior in Simulink	2-2
Create a Simulink Behavior Model	2-2
Link to an Existing Simulink Behavior Model	2-4
Export Simulink Model as Architecture	2-5

Compose an Architecture Model

- “Compose and Analyze a System” on page 1-2
- “Create an Architecture Model” on page 1-5
- “Inspect Components in Custom Views” on page 1-17
- “Analyze an Architecture Model” on page 1-21

Compose and Analyze a System

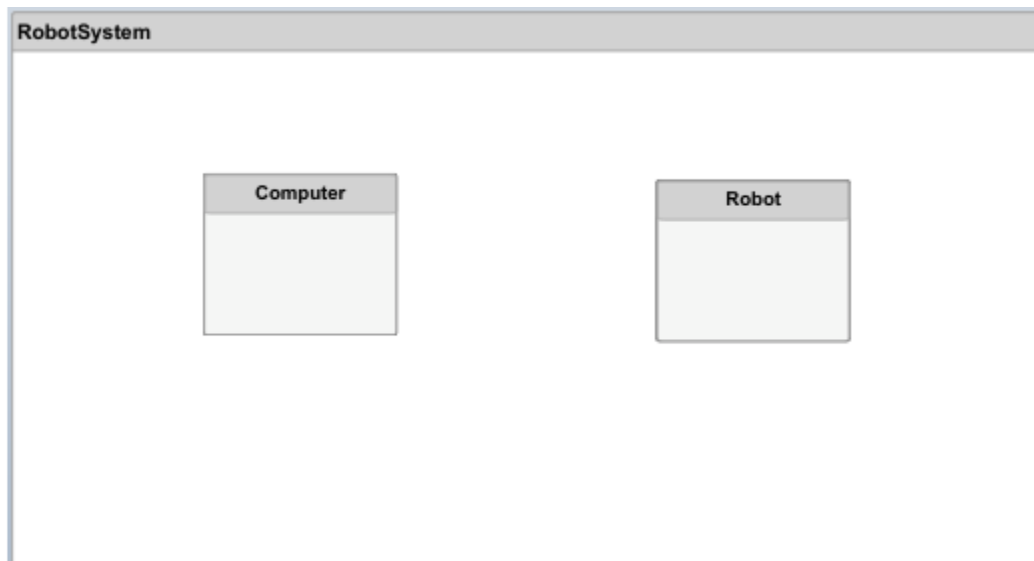
A system is a composition of different elements that serves a goal that cannot be achieved by any of the elements alone. The constituent elements of a system can be mechanical parts, electrical circuits, computer hardware, and software. A system specification consists of a description of the associated set of elements, their characteristics and properties, their interactions with each other, and the desired interaction (or interface) of the overall system with its environment.

System Composer enables you to describe systems in terms of architecture models as a combination of structural elements that are further elaborated by underlying behavioral descriptions. These descriptive models may sometimes be presented as distinct diagrams that are all kept consistent with each other.

With System Composer, you can:

- Create hierarchical models of system structure that represent functional, logical, or physical decompositions of the system through a component, port, and connector paradigm.
- Create and manage interfaces between structural architecture elements.
- Refine and elaborate requirements through workflows enabled by Simulink Requirements.
- Extend base architectural elements to create custom domain specific conceptual representations with associated properties.
- Specify component behavior in various formalisms including Simulink block diagrams, State charts in Stateflow, MATLAB and C/C++ Code.
- Perform static analysis and trade studies to optimize system architectures.
- Define custom filtered views of the system's structure for various design concerns.

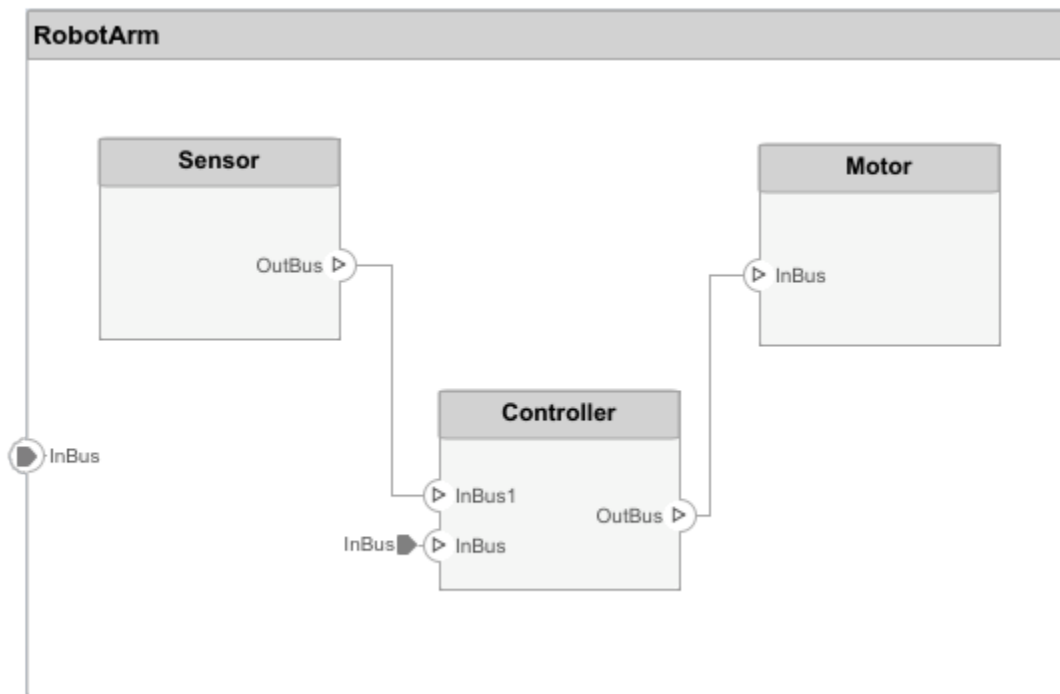
Consider a mobile robotic system where the intent is to design a computer that manages a robot wirelessly by assigning it a desired location periodically that the robot needs to reach. An architecture model of such a system could consist of two primary components: the computer and the robot. You represent them in System Composer using two component blocks.



While thinking through the initial design of the system, you start to elaborate each component further. For instance, you can define the robot as having a sensor to sense position, a controller to

manipulate the robot's position, and a motor that is actuated by the controller to reach a desired position.

You can also begin to create connectivity between the components making up the system by describing the flow of power, energy, data, or any representative information. This is done by creating ports on the components that provide or consume some desired information and connectors that bind two component ports to represent the flow of that information. You can define an interface to fully specify a connection and its associated ports. An interface can consist of multiple data elements with various dimensions, units, and data types. To enable consistency checking when connecting a port, you can also associate interfaces with unconnected ports during component design.



Requirements are integral to the system engineering process. Some are related to the functionality of the overall system and some to aspects of performance such as power, size, and weight. Decomposing high-level requirements into lower-level requirements and deriving additional requirements is an important aspect of defining the architecture of the overall system. For instance, the overall power consumption of the robot determines the requirement for the robot controller's power consumption.

To allocate and trace requirements with system elements, System Composer fully integrates with Simulink® Requirements™. To facilitate the derivation of appropriate requirements, it is sometimes necessary to analyze and specify properties (such as power) for elements of the system including components, ports, or connectors. 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 an electrical component using a stereotype. A stereotype adds properties to components, ports, and connectors. With these properties specified, you can use MATLAB to perform analysis and allocate power appropriately to all elements within the design. You can then create additional derived requirements for the designers of individual components in the system such as the controller or the sensors.

Finally, you can also begin to design the actual system's components using Simulink. You can fully specify, test, and analyze the behavior of a component in code using the Model-Based Design process based on the Simulink platform.

See Also

More About

- “Create an Architecture Model” on page 1-5
- “Inspect Components in Custom Views” on page 1-17
- “Analyze an Architecture Model” on page 1-21

Create an Architecture Model

In this section...

“Visually Represent the System” on page 1-5

“Edit Interfaces” on page 1-12

“Decompose Components” on page 1-13

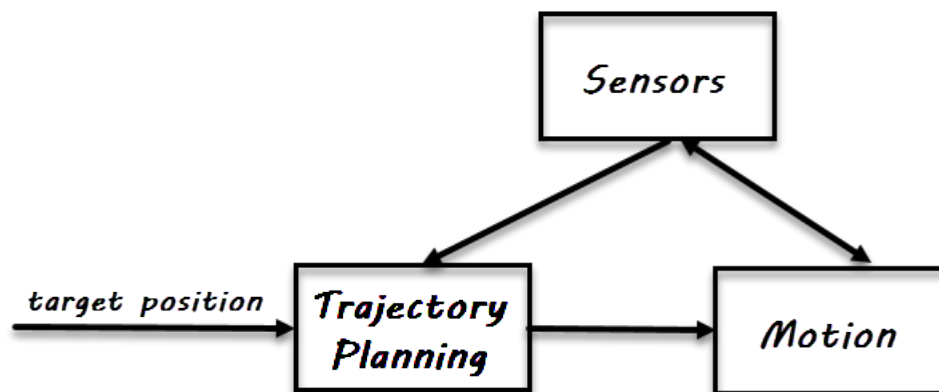
“Implement Component Behavior” on page 1-15

“Link to an Existing Simulink Behavior Model” on page 1-16

In this example, create an architecture model of a mobile robot that consists of sensors, motion, and a planning algorithm. Define the interfaces and link the requirements.

Visually Represent the System

Capture the construction of a robot arm using System Composer. The robot arm consists of the components shown in this preliminary sketch.

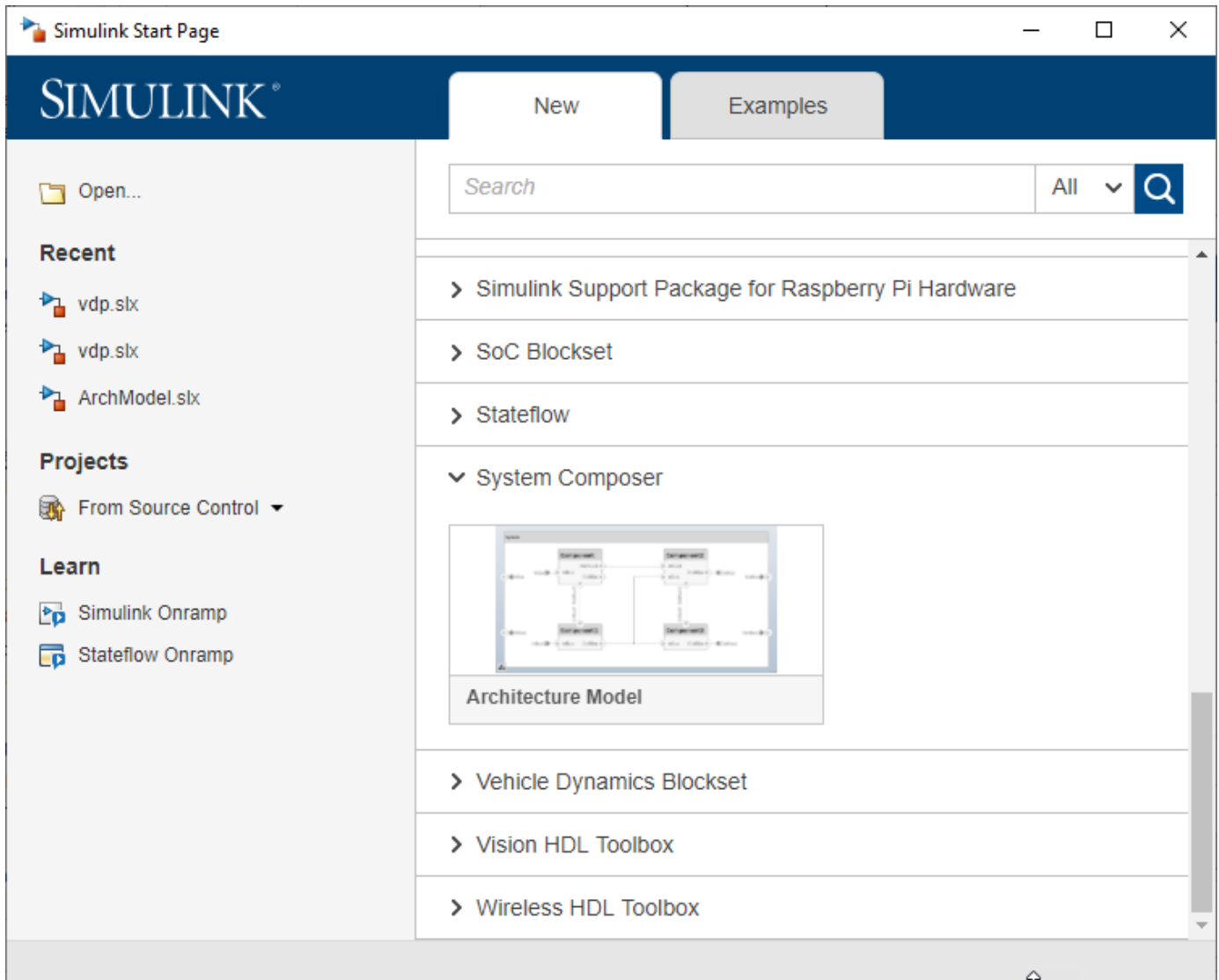


Create Architecture Model

- 1 In the MATLAB® Command Window, type
systemcomposer

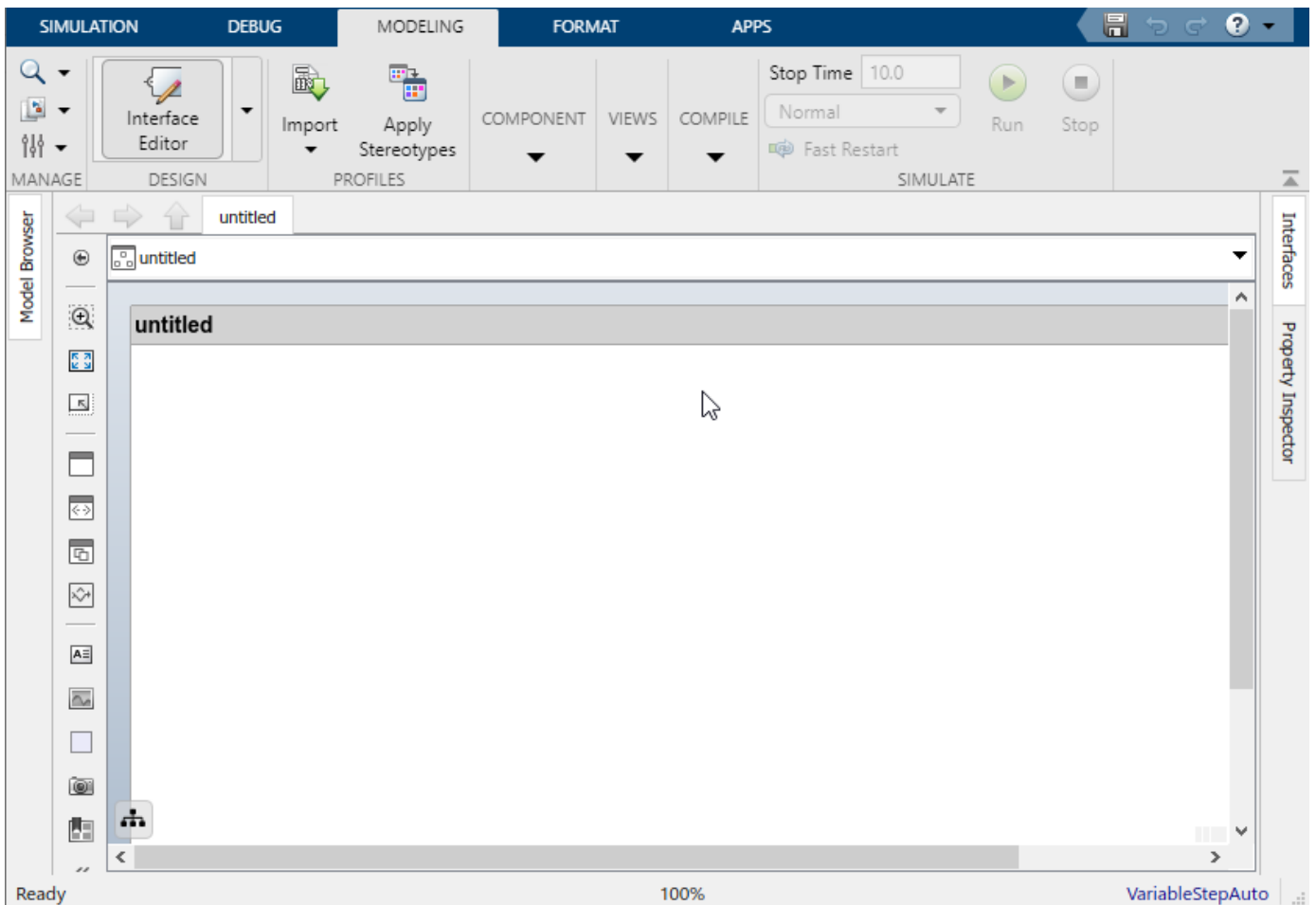
The Simulink Start Page opens to System Composer.

1 Compose an Architecture Model



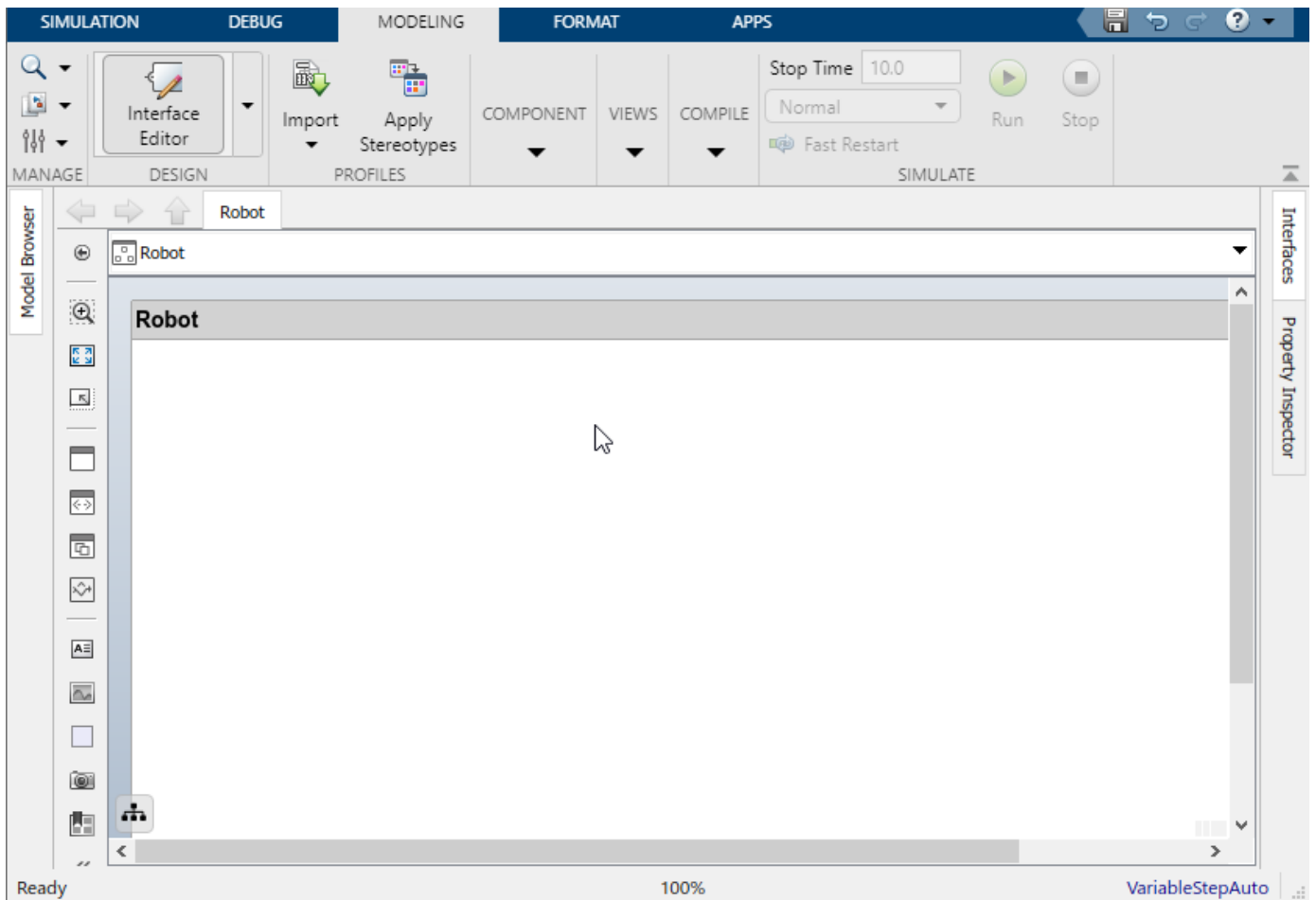
2 Click **Architecture Model**.

A composition editor window opens with a new architecture model. You can identify an architecture model by the badge in the lower left corner and the component palette on the left side.



- 3 Double-click the architecture model header and change **untitled** to **Robot**. The name of the model generally reflects the system whose architecture you are building.

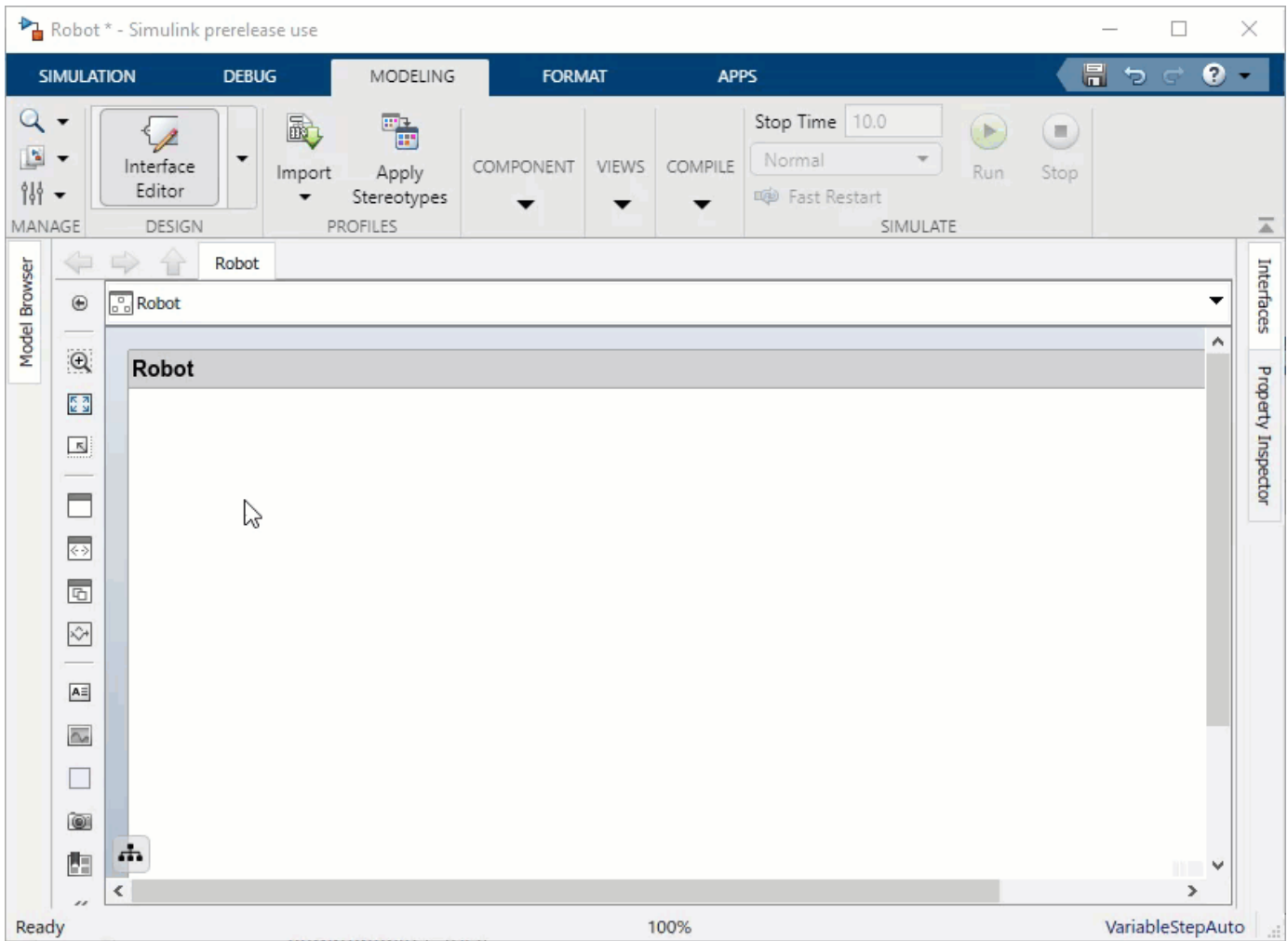
1 Compose an Architecture Model



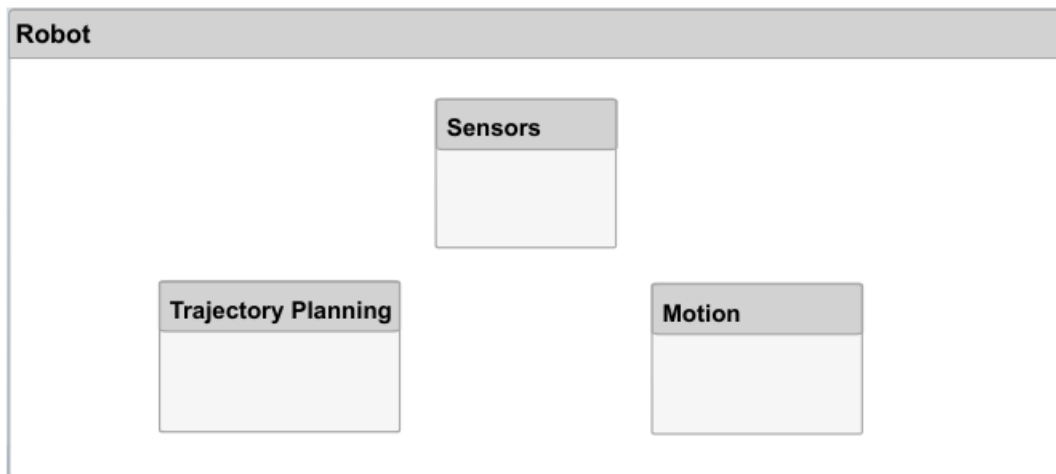
- 4 Save the model.

Draw Components



- 1 Click and drag a Component  from the left-side palette.



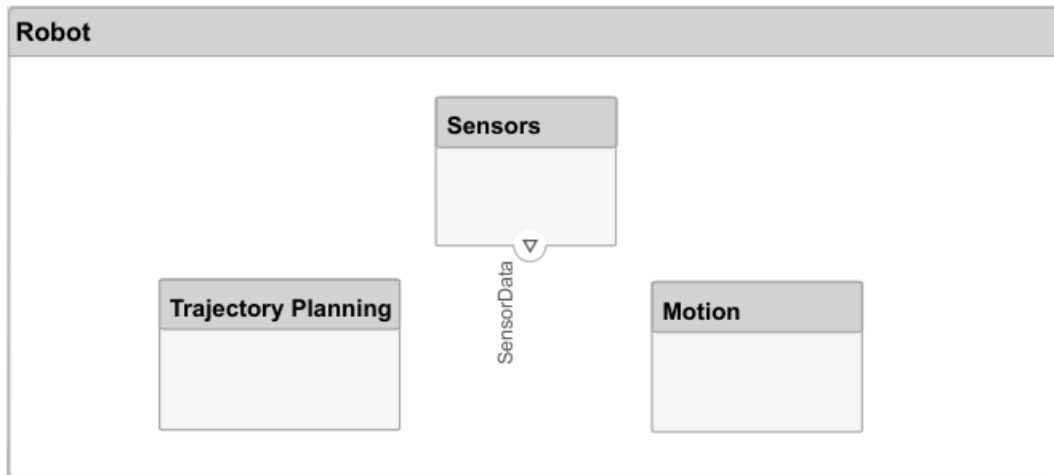
- 2 Rename the component to **Sensors**.
- 3 Add **Trajectory Planning** and **Motion** components.




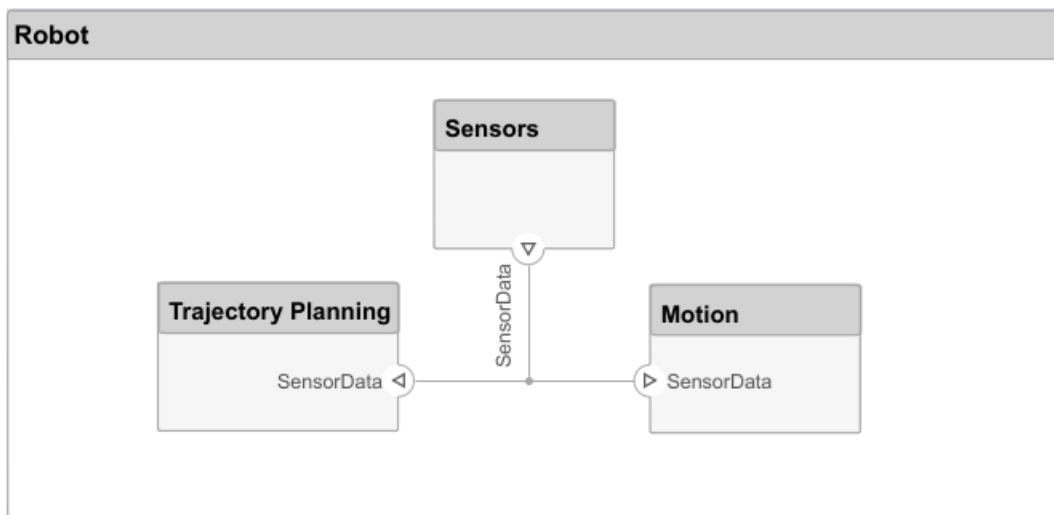
Create Ports and Connections

You can add a port to a component on any side and the port can have either an input or output direction. To create a port, pause the mouse cursor over a component side. Click and release the mouse button to view direction options. Select either  or  to select the direction. Rename the port using a name that is representative of the information that flows through that port.

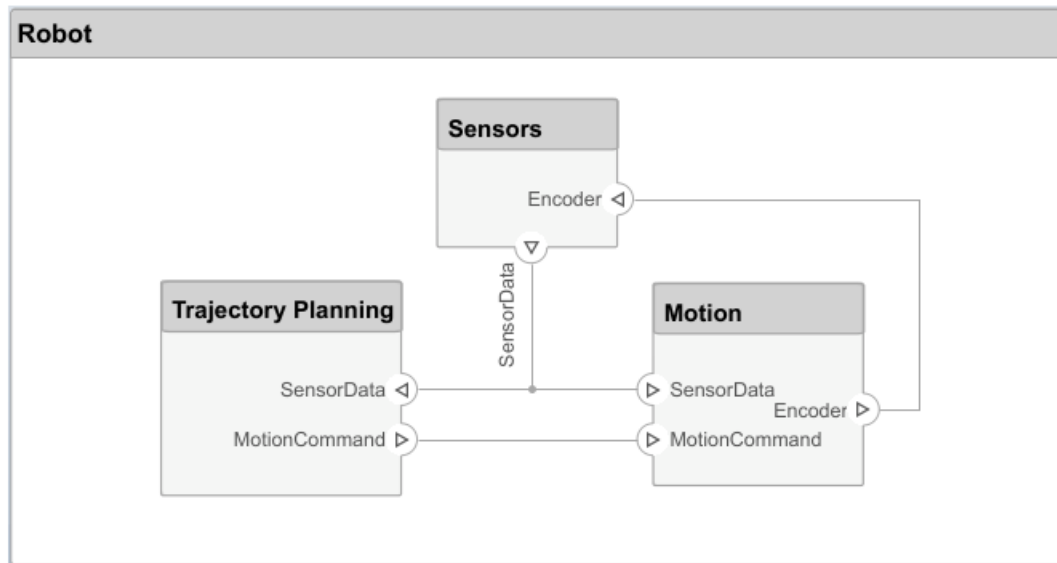
- 1 Create an output port on the bottom side of the **Sensors** component. Rename it **SensorData**.



- 2 Click and drag a line from the **SensorData** output port to the **Motion** component. When you see an input port created at the component side, release the mouse button. By default, this new port has the same name as the source port.
- 3 Hover on the corner of the **SensorData** line until you see the branch icon. . Right-click and drag a branch line to the **Trajectory Planning** component.



- 4 Complete connections as shown in the following figure.

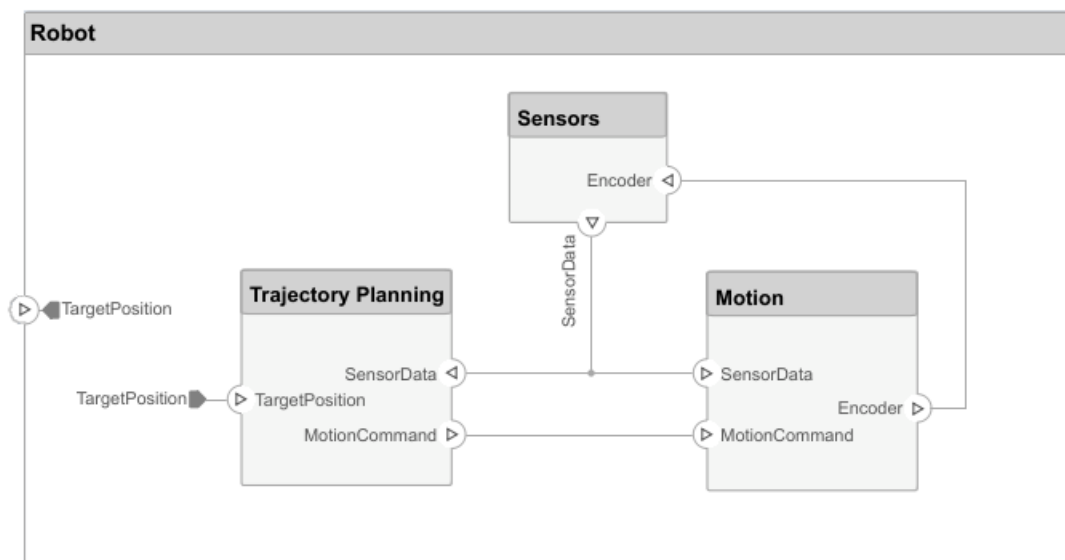


The root level of the architecture model can also have ports that describe the interaction of the system with its environment. In this example, the target position for the robot is provided by a computer external to the robot itself. Represent this relationship with an input port.

- 1 Click the left edge of the architecture model and type the port name **TargetPosition**.



- 2 Connect an architecture port to a component by dragging a line from the **TargetPosition** input port to the **Trajectory Planning** component. Connections to or from an architecture port appear as tags.



Edit Interfaces

Specify the data flow between components by configuring the data interface with data types, units, dimensions, and other attributes. An interface can be as simple as sending an integer value, but it can also 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 **Sensors** and the **Motion** components. The sensor data consists of:

- Position data from two motors
- Obstacle proximity data from two sensors
- A time stamp to capture the freshness of the data

The data has these specifications.

Name	Data Type	Unit
timestamp	double	seconds
position1 for motor 1	double	degrees
position2 for motor 2	double	degrees
distance1 for sensor 1	double	meters
direction1 for sensor 1	double	degrees
distance2 for sensor 2	double	meters
direction2 sensor 2	double	degrees

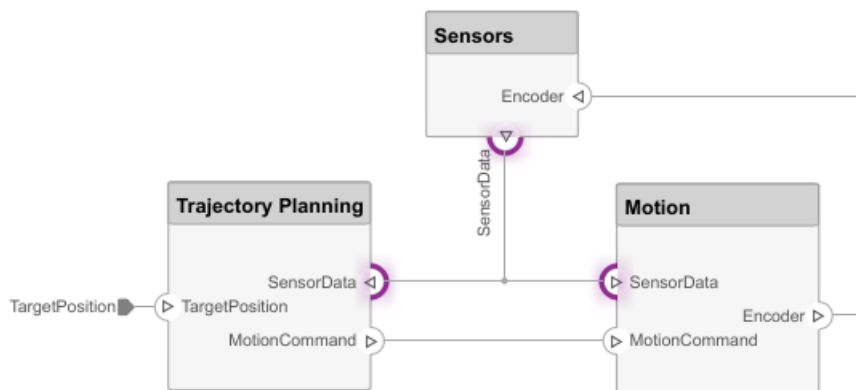
1 In the **Modeling** tab, select Interface Editor.


2 Click the  button to add an interface and name it **sensordata**.

The interface is named and defined separately from a component port and then assigned to a port.


3 Click the **SensorData** output port on the **Sensors** component. In the Interface Editor, right-click **sensordata** and select **Assign to Selected Port(s)**.

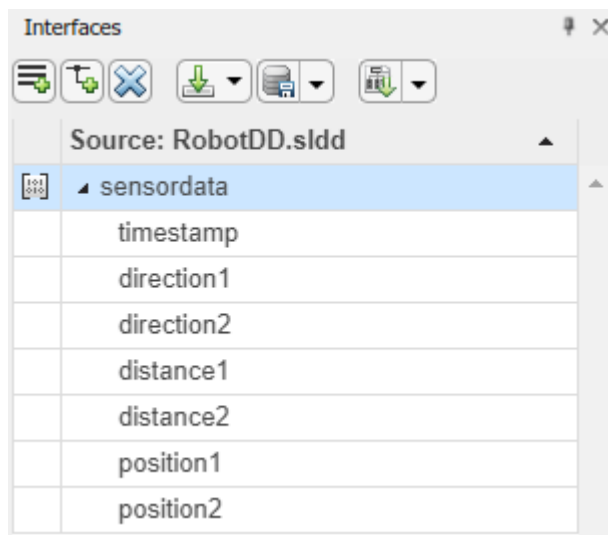
If you click **sensordata** again, the three **SensorData** ports are highlighted, indicating the ports are associated with that interface.



- 4 Add an element to the selected interface. Click the  button to add an element and name it timestamp.
- 5 Continue adding elements to the interface as specified by clicking the add element button.
- 6 Edit the properties of an interface element using the Property Inspector. Right-click an any element and select **Inspect Properties**. The Property Inspector opens above the Interfaces editor.

Click each interface element and add units as shown in the specification. Click the drop-down

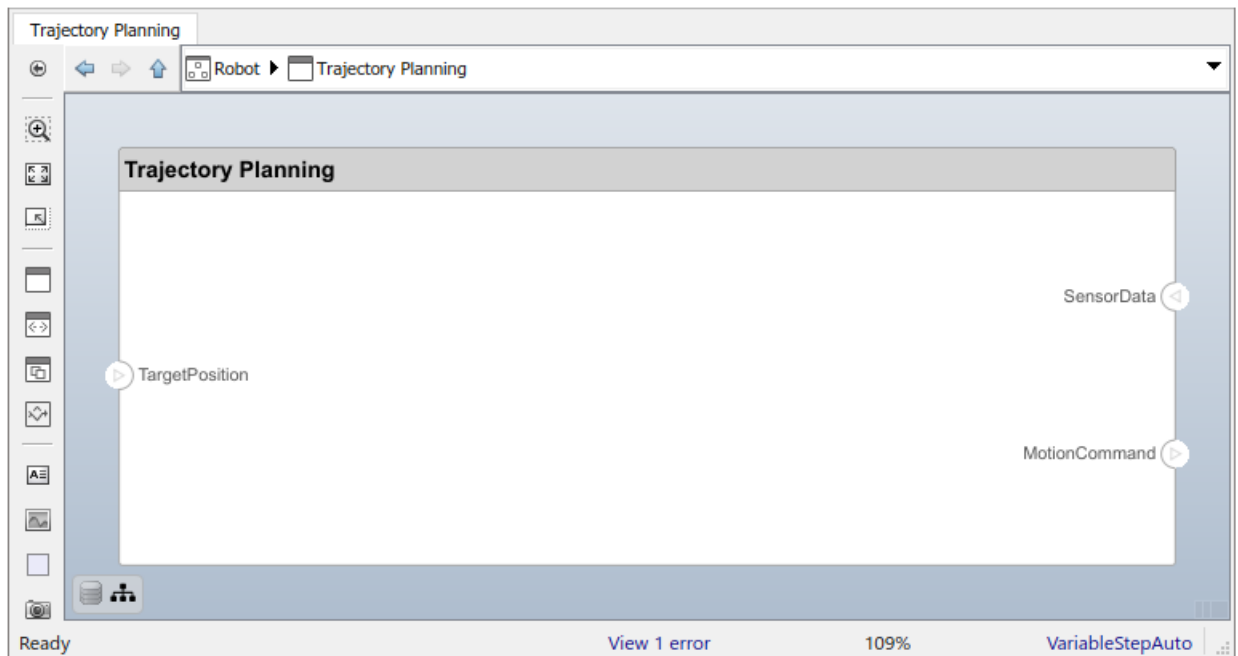
next to the  button to save the interface to a data dictionary. A data dictionary allows you to collectively manage and share a set of interfaces among interrelated models. For instance, later in the design, if you choose to model the external computer as a separate architecture model, then this model and the Robot model can share the same data dictionary. Here it is saved as RobotDD.



Decompose Components

Each component can have its own architecture. Double-click a component to decompose it into its subcomponents.

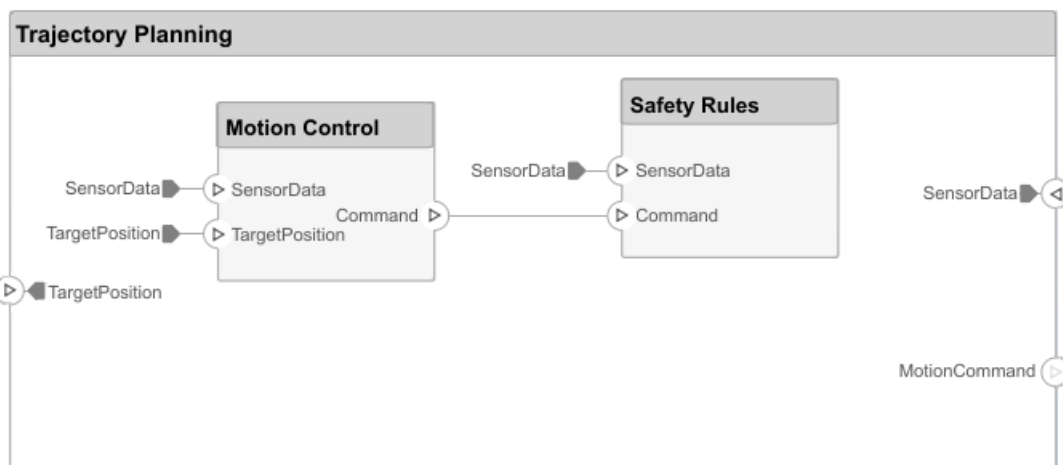
- 1 Double-click the **Trajectory Planning** component. The title or **Model Browser** indicates the position of the component in the model hierarchy.



This component first uses the motor position data that is part of the `sensorData` interface to compute the ideal position and velocity command. It then processes the obstacle distance information in the same interface to condition this motion command, according to some safety rules.

- 2 Add **Motion Control** and **Safety Rules** components as part of the **Trajectory Planning** architecture.

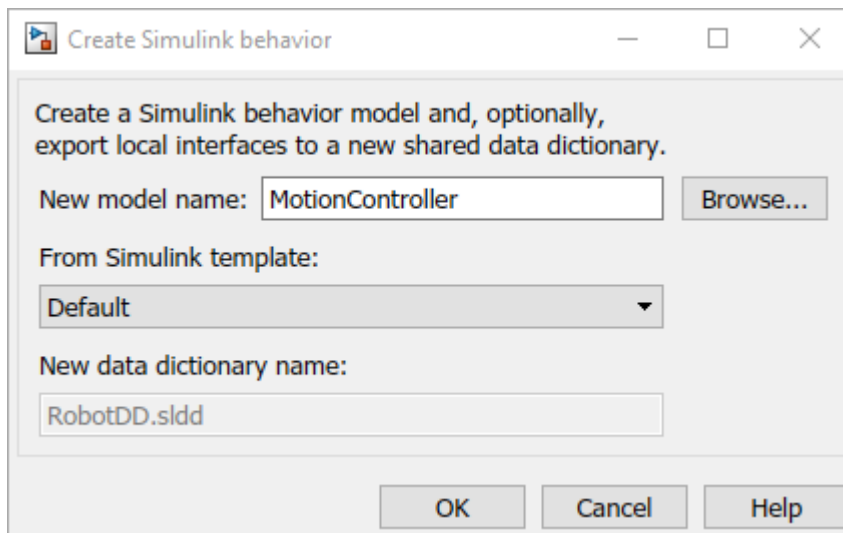
Drag the **TargetPosition** port to the **Motion Control** component. Add a **Command** output port to **Motion Control**, and then drag a line to the **Safety Rules** component. Drag lines from **SensorData** port to **Motion Control** and **Safety Rules** components.



Implement Component Behavior

If you have a component that represents a single functional unit that does not need further architectural decomposition, you can define either its intended or detailed behavior model in Simulink.

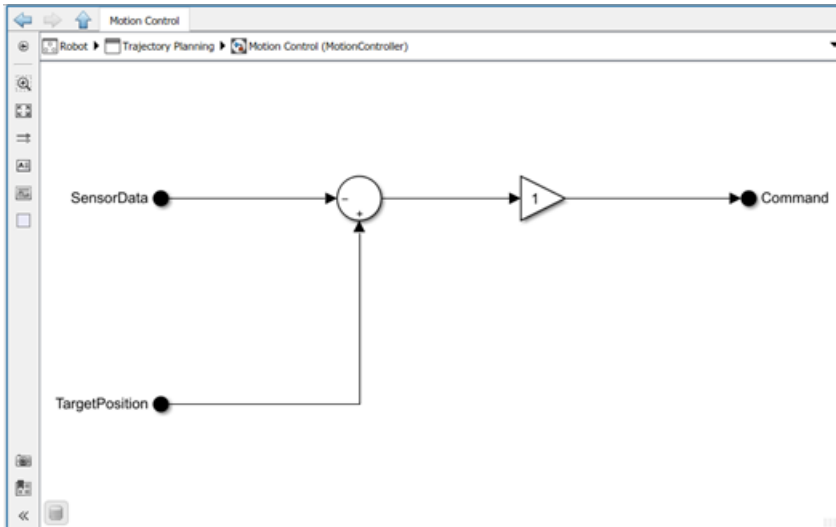
- 1 Right-click the **Motion Control** component and select **Create Simulink Behavior**.
- 2 Type the name for a new Simulink model. Select either a blank Simulink template (default) or a template that you created and click **OK**. For more information about creating your own Simulink templates, see “Create a Template from a Model” (Simulink).



This creates a new Simulink model in the current directory, converts the component to a reference component (one that refers to another model stored as a separate artifact), and adds a Simulink model badge with the name of the referenced model on the component. The Simulink behavior model is preconfigured with the ports from the component in the architecture model. You can now use Simulink to elaborate the behavior model of the Motion Control.



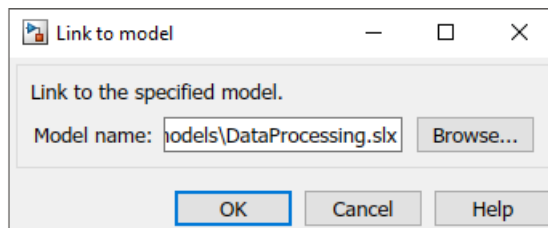
- 3 Double-click **Motion Control**. Add a Sum block to subtract SensorData from TargetPosition, and add a Gain block before connecting to the output Command to represent a simple proportional controller.



Link to an Existing Simulink Behavior Model

You can also link to an existing Simulink behavior model from a System Composer component if one is already available from a previous design.

- 1 Right-click the component and select **Link to Model**.
- 2 Enter the name of a Simulink model.



Any subcomponents and ports that are present in the component are deleted when the component links to a Simulink model. The component then is created with ports that are the same as those described in the Simulink model.

See Also

More About

- “Analyze an Architecture Model” on page 1-21
- “Create Spotlight Views”

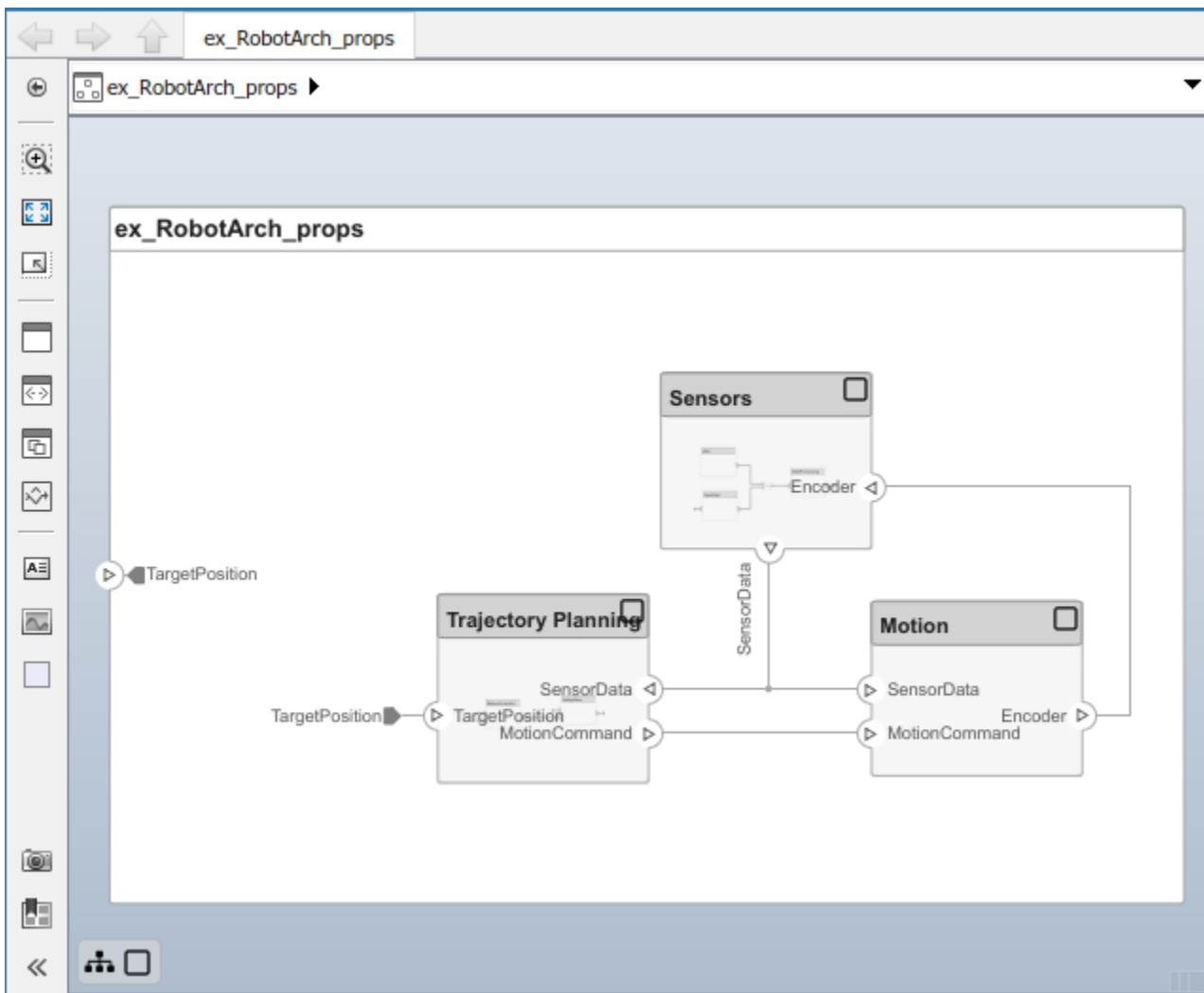
Inspect Components in Custom Views

View the hierarchy and connectivity of a component in a specialized view. Such views allow you to create simpler diagrams that only show a subset of the original model elements for a specific design activity or concern.

Open the Model

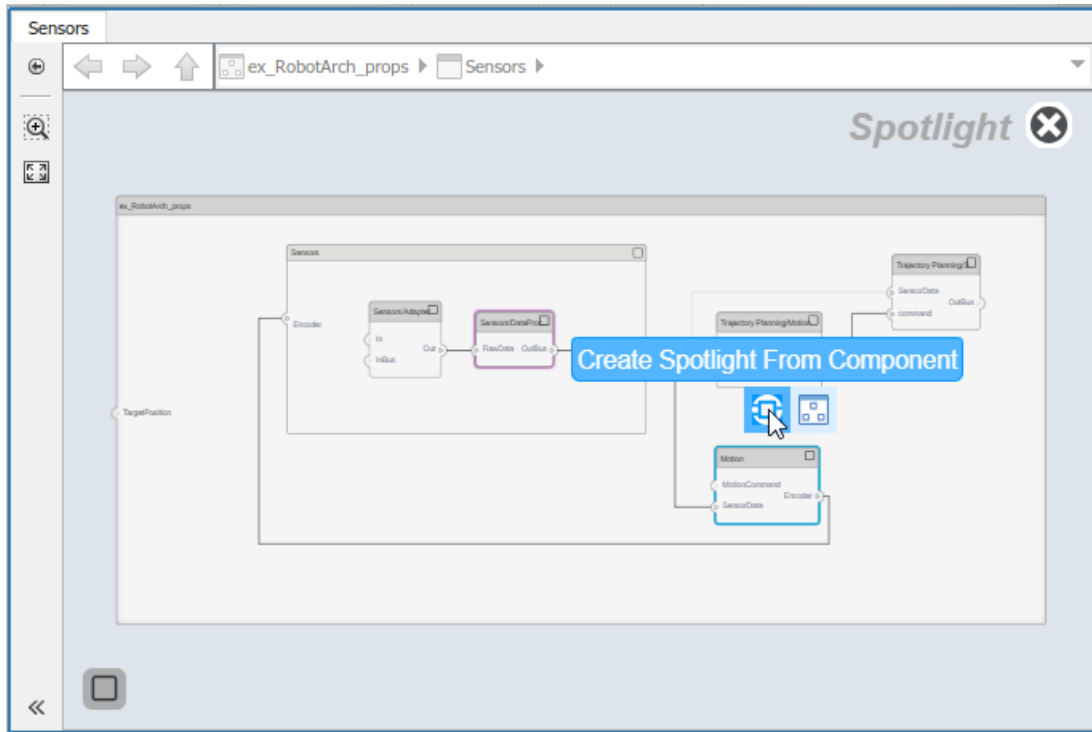
Open the `ex_RobotArch_props` model.


```
open_system('ex_RobotArch_props')
```

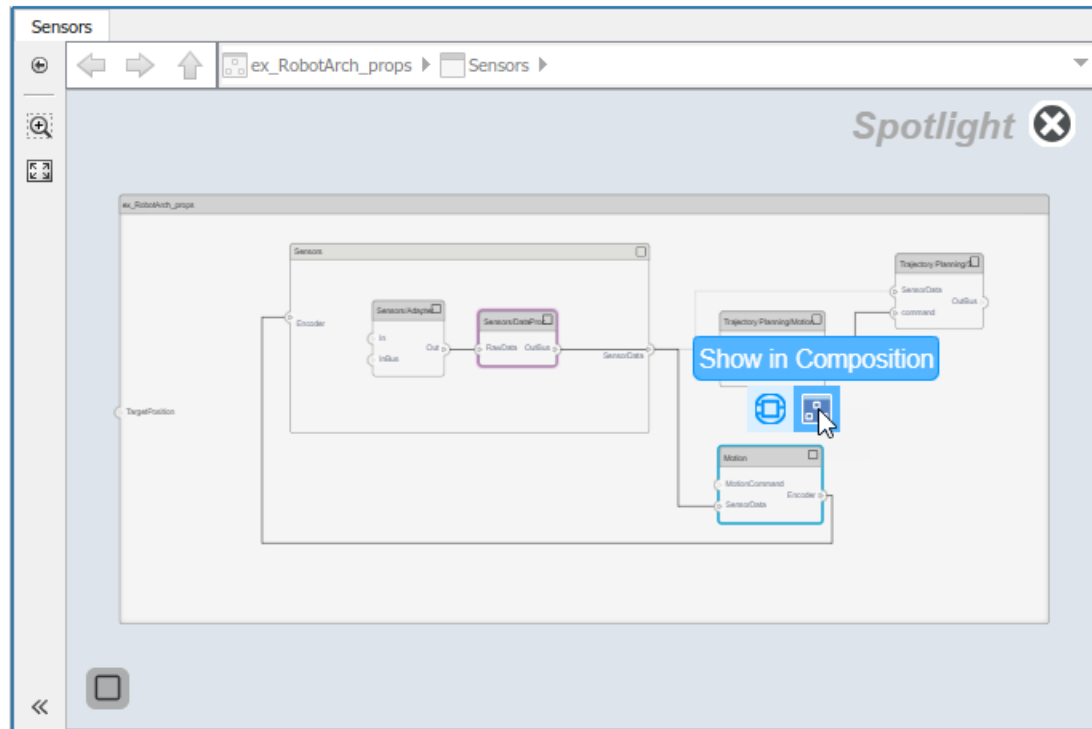



Inspect the Component and Its Connectivity

- 1 Double-click the **Sensors** component, and then select the **DataProcessing** component.
- 2 Right-click and select **Create Spotlight From Component**.



To view the architecture model at the level of a particular component, select the component and click the  icon.



- 4 Return to the architecture model view by clicking .
- 5 Close the model by closing the System Composer window.

Note More sophisticated filtering conditions can be created using the Architecture Views Editor. For details, see “Creating Architecture Views Interactively”.

See Also

More About

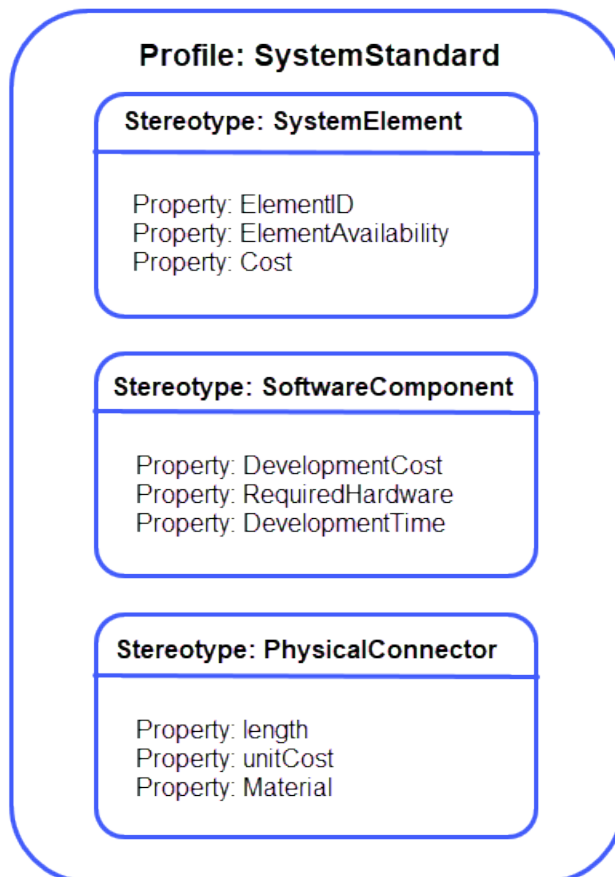
- “Analyze an Architecture Model” on page 1-21

Analyze an Architecture Model

In this section...

“Load Architecture Model Profile” on page 1-22
 “Apply Stereotypes to Model Elements” on page 1-23
 “Set Properties” on page 1-25
 “Perform an Analysis” on page 1-26

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 made applicable only to a specific element type, such as components. When a model element has a stereotype applied to it, you can specify property values as part of its architectural definition. In addition to allowing you to manage properties relevant to the system specification within the architecture model, stereotypes and associated properties also facilitate analysis of an architecture model.




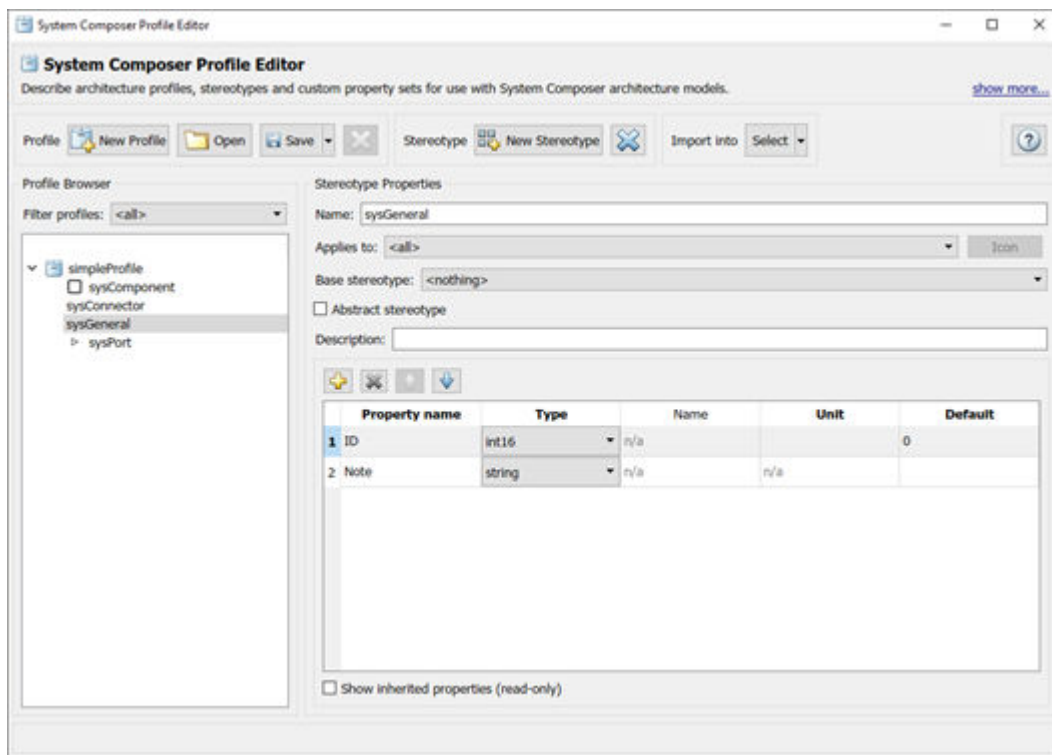
Each profile contains a set of stereotypes, and each stereotype contains a set of properties.

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

Load Architecture Model Profile

Load a profile to make stereotypes available for model elements.

- 1 Start System Composer. Enter `systemcomposer` at the MATLAB command line.
- 2 In the **Modeling** tab, select **Import** and then from the drop-down, select **Import** .
- 3 Browse to the examples folder: `<matlabroot>\toolbox\systemcomposer\examples`.
- 4 Select `simpleProfile`.
- 5 From the toolbar, click **Import** and select **Edit** to open the Profile Editor.



In the profile, observe 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.

- `sysGeneral` is a general stereotype, applicable to all element types, that enables adding generic properties such as a `Note`, which project members can use to track any issues with the element.
- `sysComponent` applies only to components, and includes properties such as weight and cost that contribute to the total weight and cost specifications of the robot system.
- `sysConnector` stereotype applies to connectors, and includes price and weight properties defined per meter of length (assuming a physical connector like a wire). These properties help compute the total weight and cost of the design in this particular example.

Note You can add a stereotype icon to all component-level stereotypes. These are your choices:

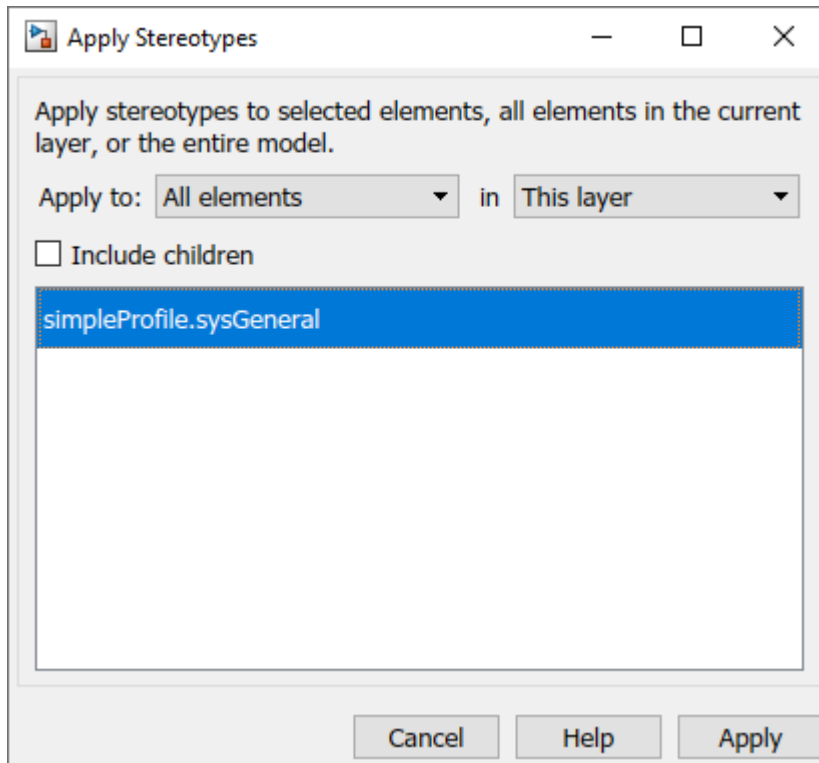


Apply Stereotypes to Model Elements

Add custom properties to a model element by applying a stereotype from a loaded profile. This procedure uses the model `ex_RobotArch`.

- 1 Open the **Sensors** component.
- 2 On the **Modeling** tab, select **Apply Stereotypes**.
- 3 In the Apply Stereotypes dialog box and from the **Apply to** list, select **All elements**. From the **in** list, select **This layer**.

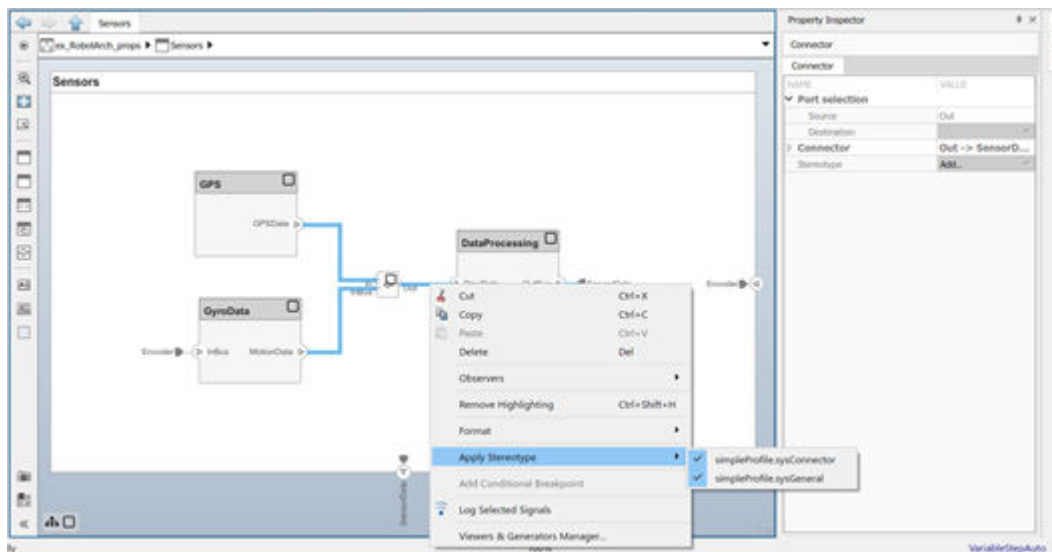
In the list of available stereotypes, select `simpleProfile.sysGeneral`.



- 4 Click **Apply** and close the window to exit the dialog box.
- 5 Select the **GPS** component. Right-click and select **Apply Stereotype**. Check that `simpleProfile.sysComponent` is selected.

Note The `sysComponent` stereotype is used for managing physical properties and cost.

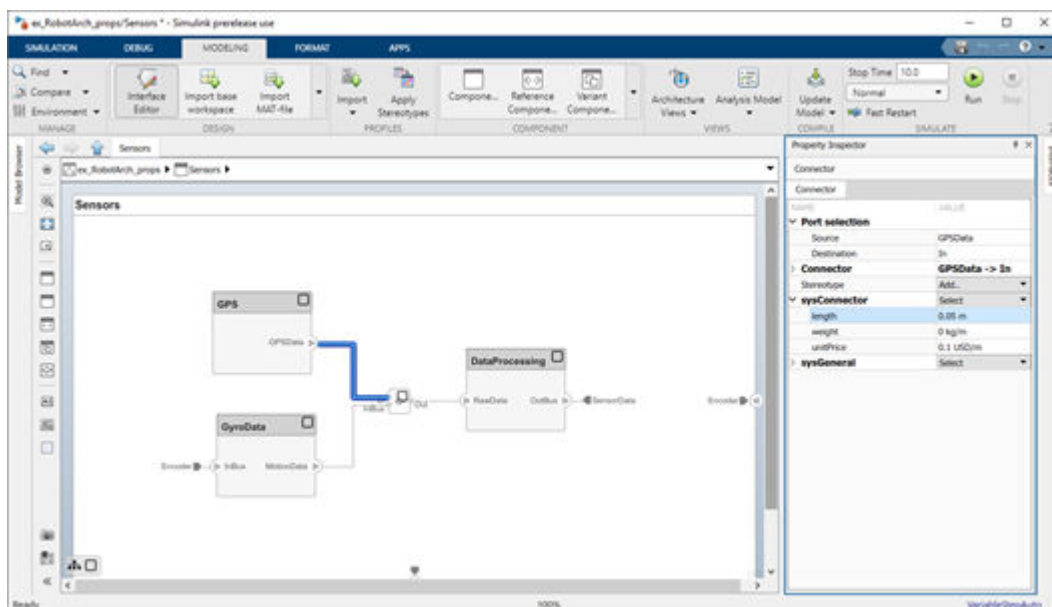
- 6 Navigate to top of the model. Check that the `sysComponent` stereotype is applied to the **Sensors** and **Trajectory Planning** components, and the top-level architecture model. Right-click each component or a space in the top-level, and select **Apply Stereotype** to ensure `simpleProfile.sysComponent` is selected.
- 7 Check that the `sysConnector` stereotype is applied to all connectors in the Sensors layer, the Trajectory Planning layer, and the top model layer. Press and hold **Shift** to select multiple connectors. Right-click the selection, click **Apply Stereotype**, and check that the stereotype is selected.



Set Properties

Set the property values to enable cost analysis. Follow this example for the GPS module.

- 1 In the **Sensors** component, select the **GPS** component.
- 2 Open the Property Inspector. Click the drop-down in the **Design** section of the toolbar and select **Property Inspector**.
- 3 Expand the **sysComponent** stereotype to see the properties.
- 4 Set **unitPrice** to 5 and press **Enter**.
- 5 Select the **GPSData** port connector. Check that **length** is set to 0.05 and **unitPrice** to 0.1.



- 6** Complete the model using 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.

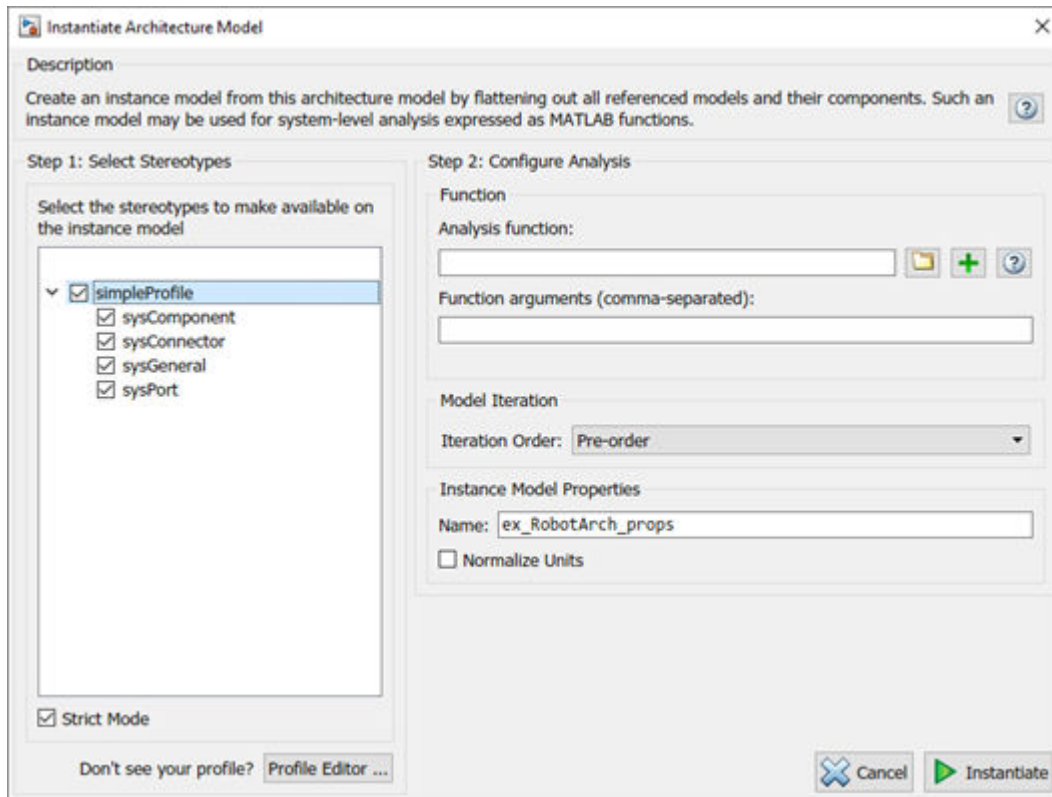
Layer	Element	Property	Value
Top layer	Encoder connector	length	0.5
		unitPrice	0.1
	SensorData connector	length	0.6
		unitPrice	0.2
	MotionCommand connector	length	0.5
		unitPrice	0.2
	Sensors component	unitPrice	5
	Trajectory Planning component	unitPrice	500
Motion component	unitPrice	750	
Sensors 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	

- 7** Save the model as `ex_RobotArch_props.slx`.


Perform an Analysis

Analyze the total cost for all components in the robot model.

- 1** On the **Modeling** tab and in the **Views** section, select **Analysis Model**, and then from the drop-down list select **Analysis Model**.



- 2 Add an analysis function. In the Analysis function box, enter the function name

`ex_RobotArch_analysis` without an extension, and then click . A MATLAB function file is created and saved with the name `ex_RobotArch_analysis.m`.

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 as a total of its own cost and the cost of all of its child components.

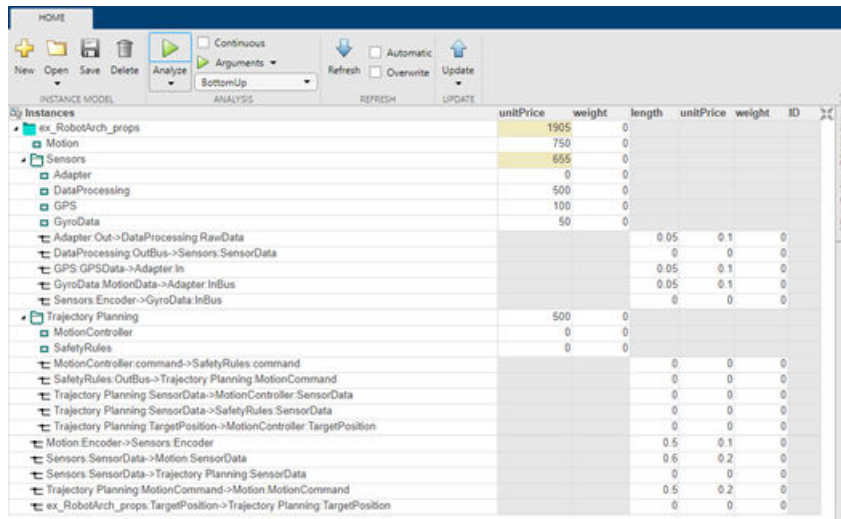
```
function ex_RobotArch_analysis(instance,varargin)

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
        if (child.hasValue('sysConnector.unitPrice'))
            unitPrice = child.getValue("sysConnector.unitPrice");
            length = child.getValue("sysConnector.length");
            sysComponent_unitPrice = unitPrice*length + sysComponent_unitPrice;
        end
    end
    instance.setValue("sysComponent.unitPrice",sysComponent_unitPrice)
end
```

- 3 Return to the Instantiate Architecture Model screen and click **Instantiate**. The Analysis Viewer shows the properties of each model element. The default values for the start of the Analysis are taken from the property values you entered when you attached the stereotype to the model and edited their values.

- 4 In the **Analysis** section, 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.



The screenshot shows the Analysis Viewer interface with a table of instance costs. The table has columns for unitPrice, weight, length, unitPrice, weight, and ID. The data is organized into a tree structure under the 'Instances' folder, with 'ex_RobotArch_props' at the top. The table shows the following data:

Instance	unitPrice	weight	length	unitPrice	weight	ID
ex_RobotArch_props	1905	0				
Motion	750	0				
Sensors	655	0				
Adapter	0	0				
DataProcessing	500	0				
GPS	100	0				
GyroData	50	0				
Adapter Out->DataProcessing RawData			0.05	0.1	0	
DataProcessing OutBus->Sensors SensorData			0	0	0	
GPS GPSData->Adapter In			0.05	0.1	0	
GyroData MotionData->Adapter InBus			0.05	0.1	0	
Sensors Encoder->GyroData InBus			0	0	0	
Trajectory Planning	500	0				
MotionController	0	0				
SafetyRules	0	0				
MotionController command->SafetyRules command			0	0	0	
SafetyRules OutBus->Trajectory Planning MotionCommand			0	0	0	
Trajectory Planning SensorData->MotionController SensorData			0	0	0	
Trajectory Planning SensorData->SafetyRules SensorData			0	0	0	
Trajectory Planning TargetPosition->MotionController TargetPosition			0	0	0	
Motion Encoder->Sensors Encoder			0.5	0.1	0	
Sensors SensorData->Motion SensorData			0.6	0.2	0	
Sensors SensorData->Trajectory Planning SensorData			0	0	0	
Trajectory Planning MotionCommand->Motion MotionCommand			0.5	0.2	0	
ex_RobotArch_props.TargetPosition->Trajectory Planning TargetPosition			0	0	0	

See Also

More About

- “Create an Architecture Model” on page 1-5
- “Define Profiles and Stereotypes”

Refactor a Simulink Model as a Composition

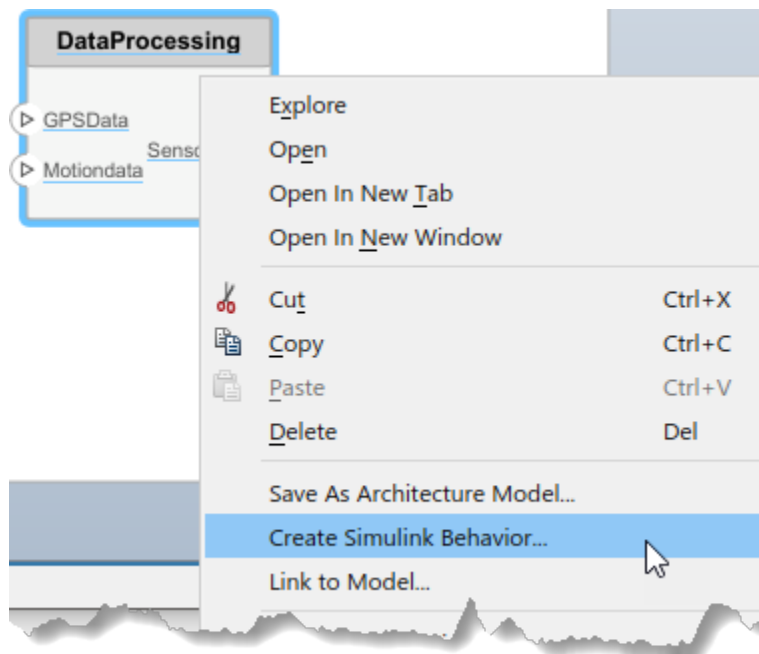
- “Implement Component Behavior in Simulink” on page 2-2
- “Export Simulink Model as Architecture” on page 2-5

Implement Component Behavior in Simulink

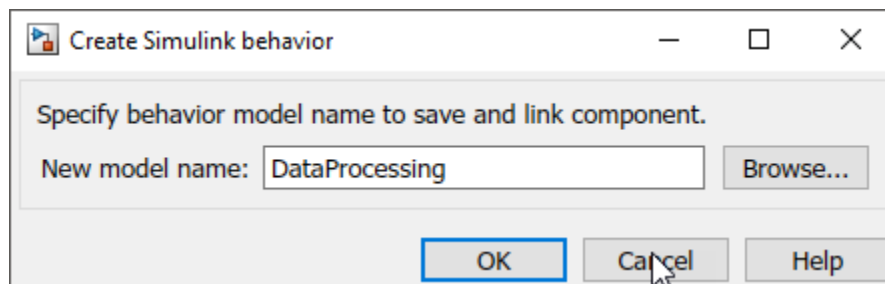
Create and use Simulink models to specify component behavior.

Create a Simulink Behavior Model

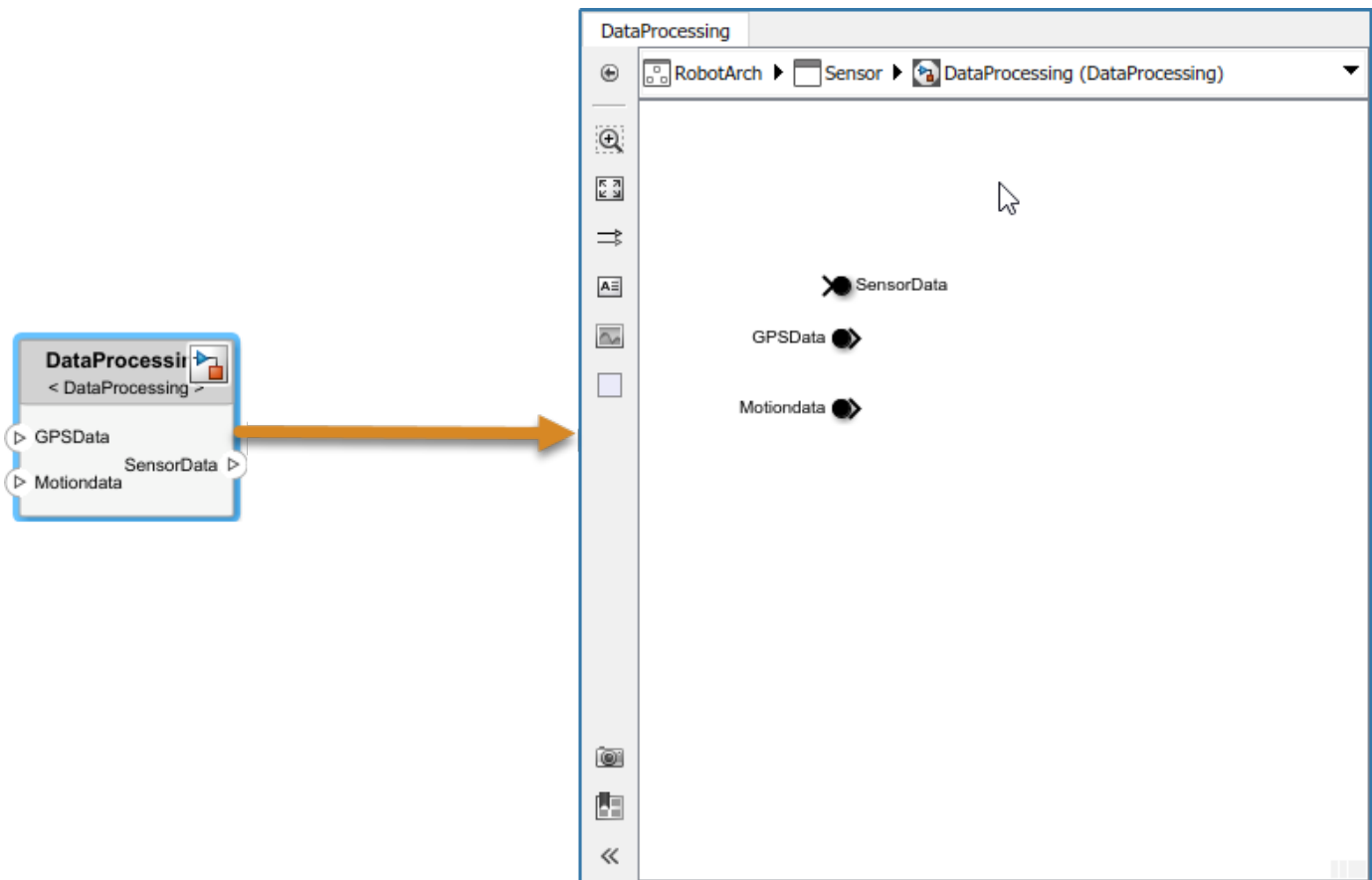
When a component reaches the level of abstraction suitable for design, you can design its behavior using Model-Based Design or specify its intended behavioral model using a Stateflow® chart that can serve as a guide for detailed implementation in Simulink. Right-click the component and select **Create Simulink Behavior**.



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

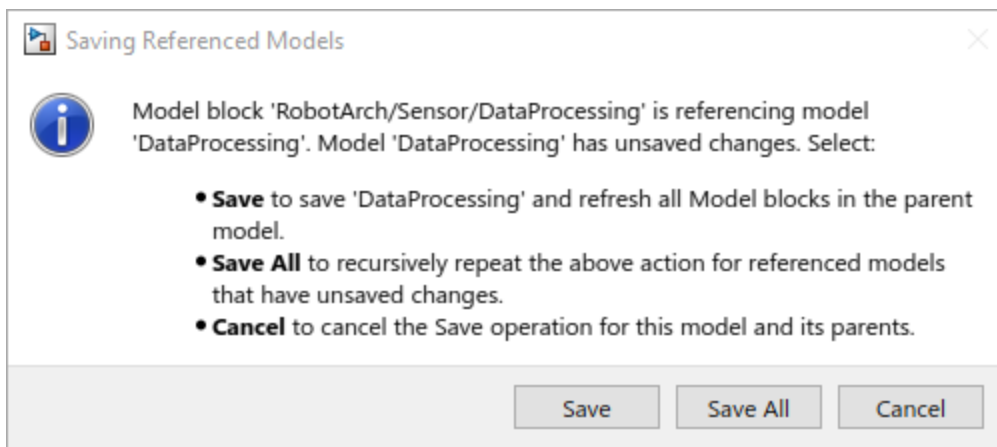


- A new Simulink model with the provided name is created. The root level ports of the Simulink model are preconfigured from the ports of the component in the architecture model.
- The component in the architecture model is linked to the Simulink model. The Simulink icon on the component indicates a Simulink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



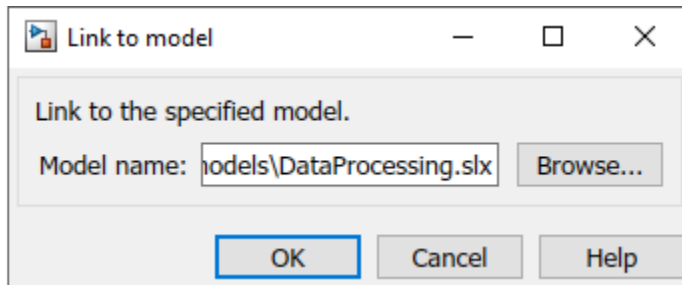
You can now go on to provide a specific algorithm implementation in the referenced Simulink model. Adding root-level ports in the Simulink model creates additional ports on the referencing component.

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 unsaved Simulink behavior models it references must also be saved, and all linked components updated.



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.

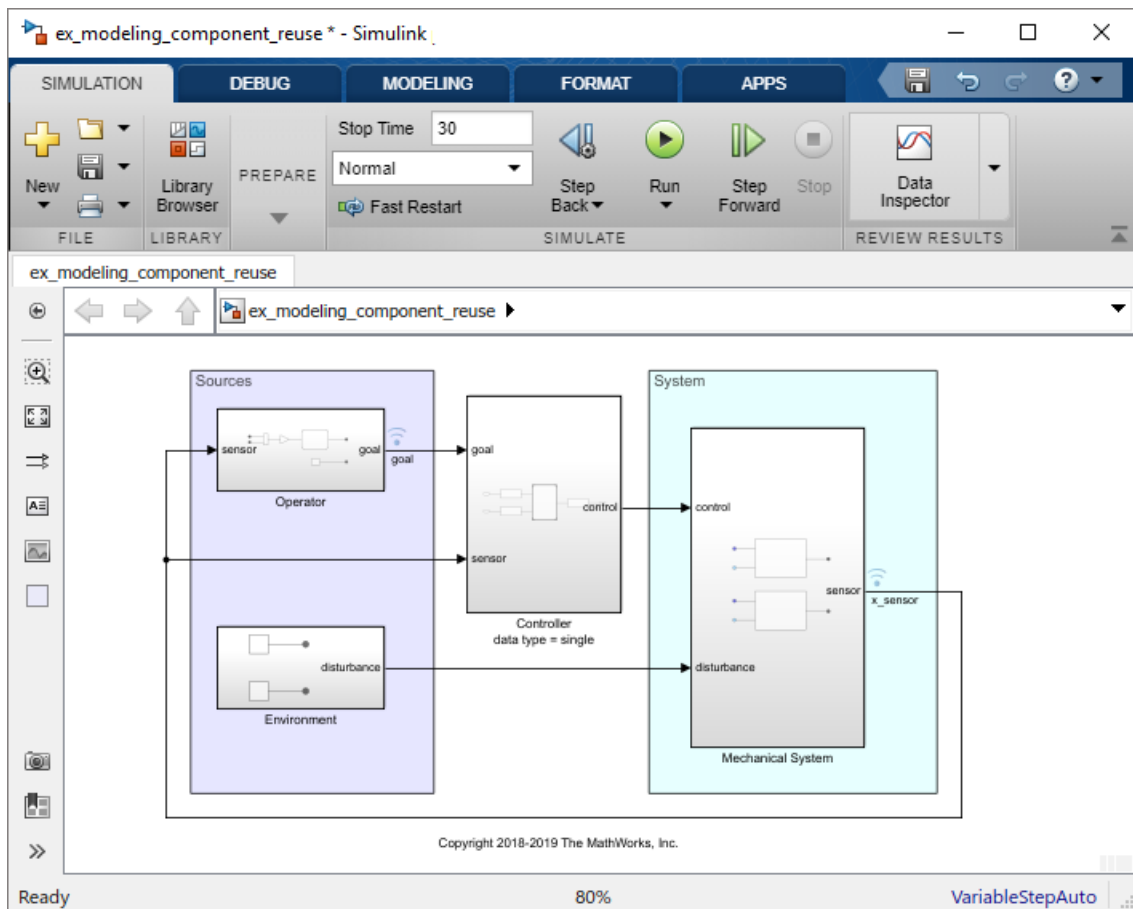


Any subcomponents and ports that are present in the components are deleted when the component links to a Simulink model and the component derives its ports from the root ports specified in the Simulink model.

Export Simulink Model as Architecture

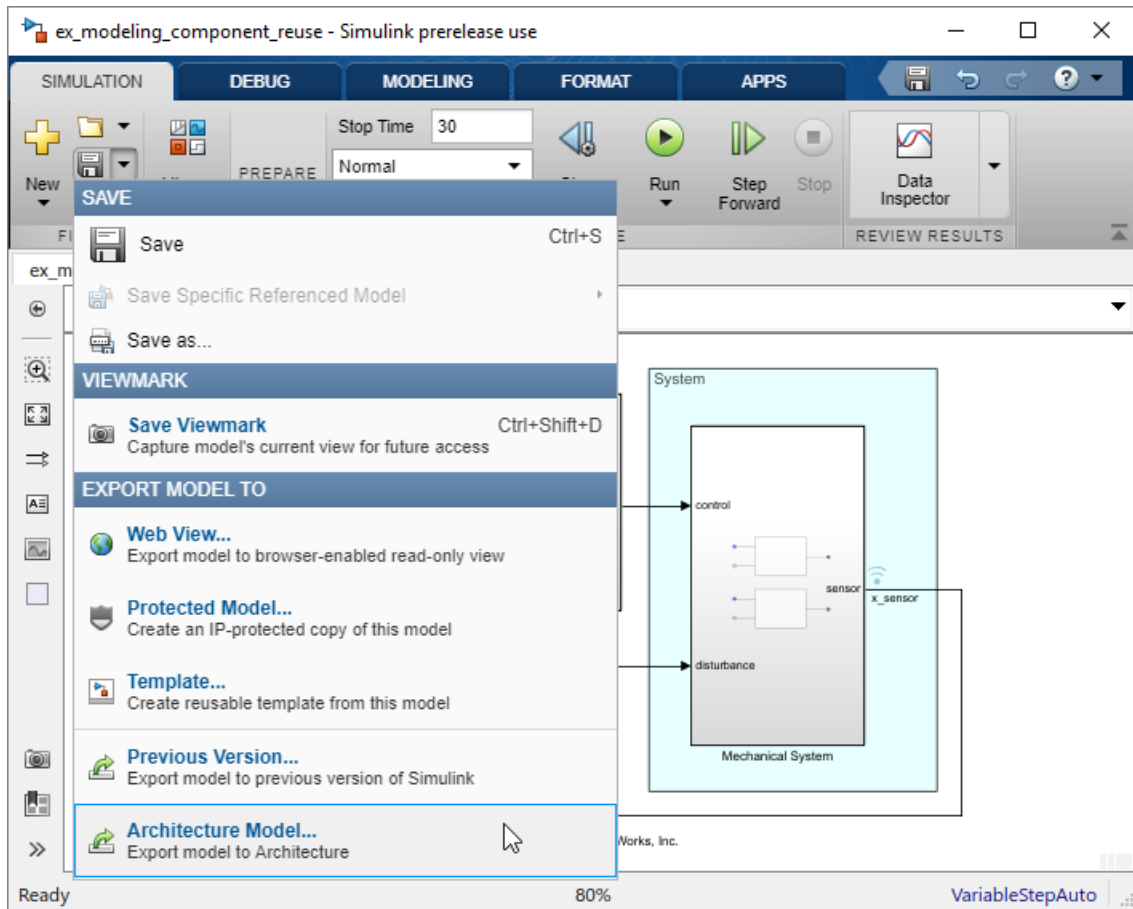
If you have an existing Simulink model of a system, you can extract an architecture model for analysis purposes or as a starting point for creating a new Architecture design in System Composer. Model and subsystem blocks, as well as all ports in a Simulink model represent architectural constructs, while 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 from the created components.

- 1 Open an example model with the MATLAB command `open_system('ex_modeling_component_reuse')`.

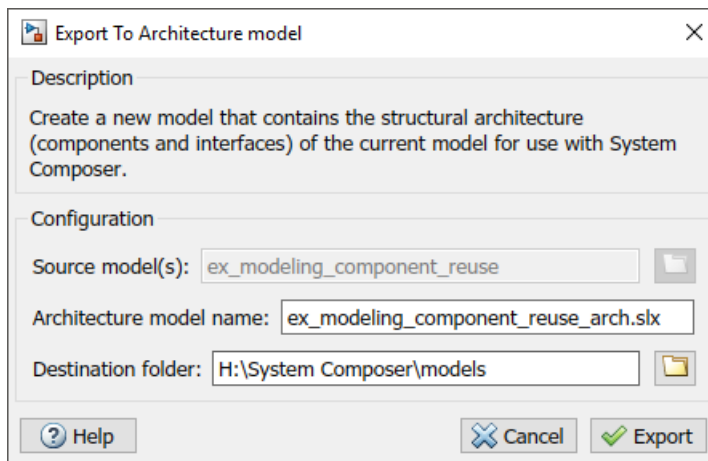


- 2 On the **Simulation** tab, click the **Save** arrow. From the **Export Model To** list, select **Architecture Model**.

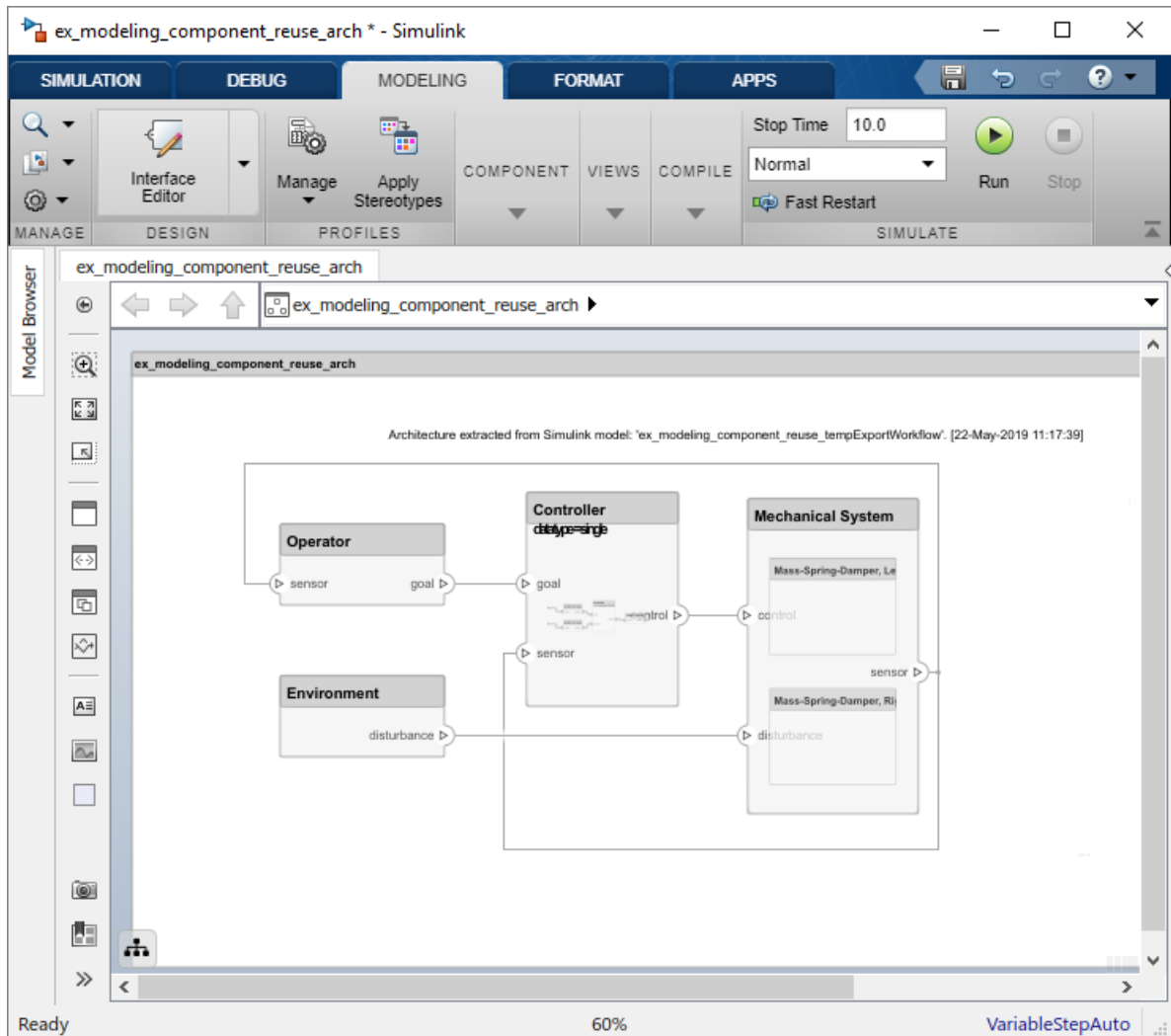
2 Refactor a Simulink Model as a Composition



- 3 Provide a name and path for the architecture model.



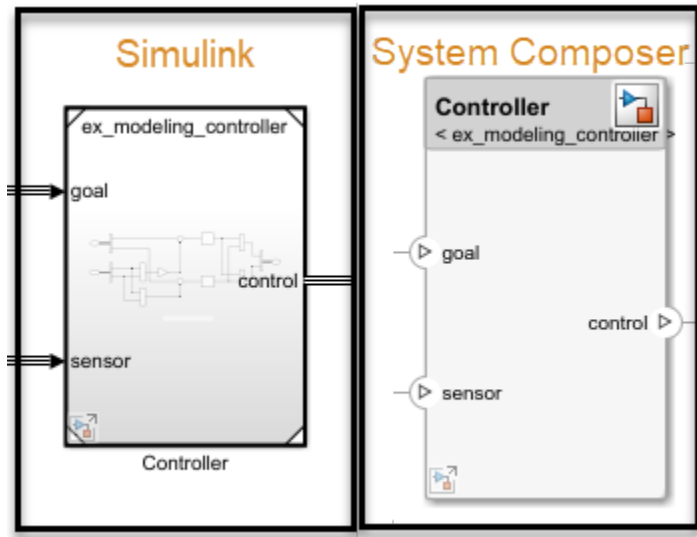
- 4 Click **Export**. A System Composer Editor window opens with an architecture model corresponding to the Simulink Model.



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.

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 and properties for model elements, and then perform analysis on the model as described previously.

See Also

More About

- “Implement Component Behavior” on page 1-15