

System Composer™

Getting Started Guide



MATLAB® & SIMULINK®

R2023a



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 2019–2023 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)
September 2020	Online only	Revised for Version 1.3 (Release 2020b)
March 2021	Online only	Revised for Version 2.0 (Release 2021a)
September 2021	Online only	Revised for Version 2.1 (Release 2021b)
March 2022	Online only	Revised for Version 2.2 (Release 2022a)
September 2022	Online only	Revised for Version 2.3 (Release 2022b)
March 2023	Online only	Revised for Version 2.4 (Release 2023a)

Product Overview

1

System Composer Product Description	1-2
--	------------

Compose an Architecture Model

2

Compose and Analyze Systems Using Architecture Models	2-2
Create Architecture Model with Interfaces and Requirement Links	2-5
Visually Represent System	2-5
Edit Data Interfaces	2-13
Decompose Components	2-15
Robot Arm Architecture Model	2-16
Manage Requirement Links	2-17
Extend Architectural Design Using Stereotypes	2-19
Mobile Robot Architecture Model	2-19
Load Architecture Model Profile	2-20
Apply Stereotypes to Model Elements	2-22
Set Properties	2-23
Analyze Architecture Model with Analysis Function	2-25
Mobile Robot Architecture Model with Properties	2-25
Perform Analysis	2-26
Inspect Components in Custom Architecture Views	2-30
Mobile Robot Architecture Model with Properties	2-30
Create Spotlight Views from Components	2-31
Create Filtered Architecture View	2-32
Implement Behaviors for Architecture Model Simulation	2-37
Robot Arm Architecture Model	2-37
Reference Simulink Behavior Model in Component	2-38
Add Stateflow Chart Behavior to Component	2-41
Design Software Architecture in Component	2-42
Represent System Interaction Using Sequence Diagrams	2-44

System Composer Concepts	3-2
Author Architecture Models	3-2
Manage Variants	3-4
Manage Interfaces	3-4
Author Physical Models	3-7
Extend Architectural Elements	3-9
Manage and Verify Requirements	3-11
Allocate Architecture Models	3-14
Create Custom Views	3-15
Analyze Architecture Models	3-17
Author Sequence Diagrams	3-18
Author Model Behavior	3-20
Design Software Architectures	3-22

Product Overview

System Composer Product Description

Design and analyze system and software architectures

System Composer enables the specification and analysis of architectures for model-based systems engineering and software architecture modeling. With System Composer, you allocate requirements while refining an architecture model that can then be designed and simulated in Simulink®.

Architecture models consisting of components and interfaces can be authored directly, imported from other tools, or populated from the architectural elements of Simulink designs. You can describe your system using multiple architecture models and establish direct relationships between them via model-to-model allocations. Behaviors can be captured and simulated in sequence diagrams, state charts, or Simulink models. You can define and simulate the execution order of component functions and generate code from your software and AUTOSAR architecture models (with Simulink and Embedded Coder®, including AUTOSAR Blockset for AUTOSAR workflows).

To investigate specific design or analysis concerns, you can create custom live views of the model. Architecture models can be used to analyze requirements, capture properties via stereotyping, perform trade studies, and produce specifications and interface control documents (ICDs).

Compose an Architecture Model

- “Compose and Analyze Systems Using Architecture Models” on page 2-2
- “Create Architecture Model with Interfaces and Requirement Links” on page 2-5
- “Extend Architectural Design Using Stereotypes” on page 2-19
- “Analyze Architecture Model with Analysis Function” on page 2-25
- “Inspect Components in Custom Architecture Views” on page 2-30
- “Implement Behaviors for Architecture Model Simulation” on page 2-37

Compose and Analyze Systems Using Architecture Models

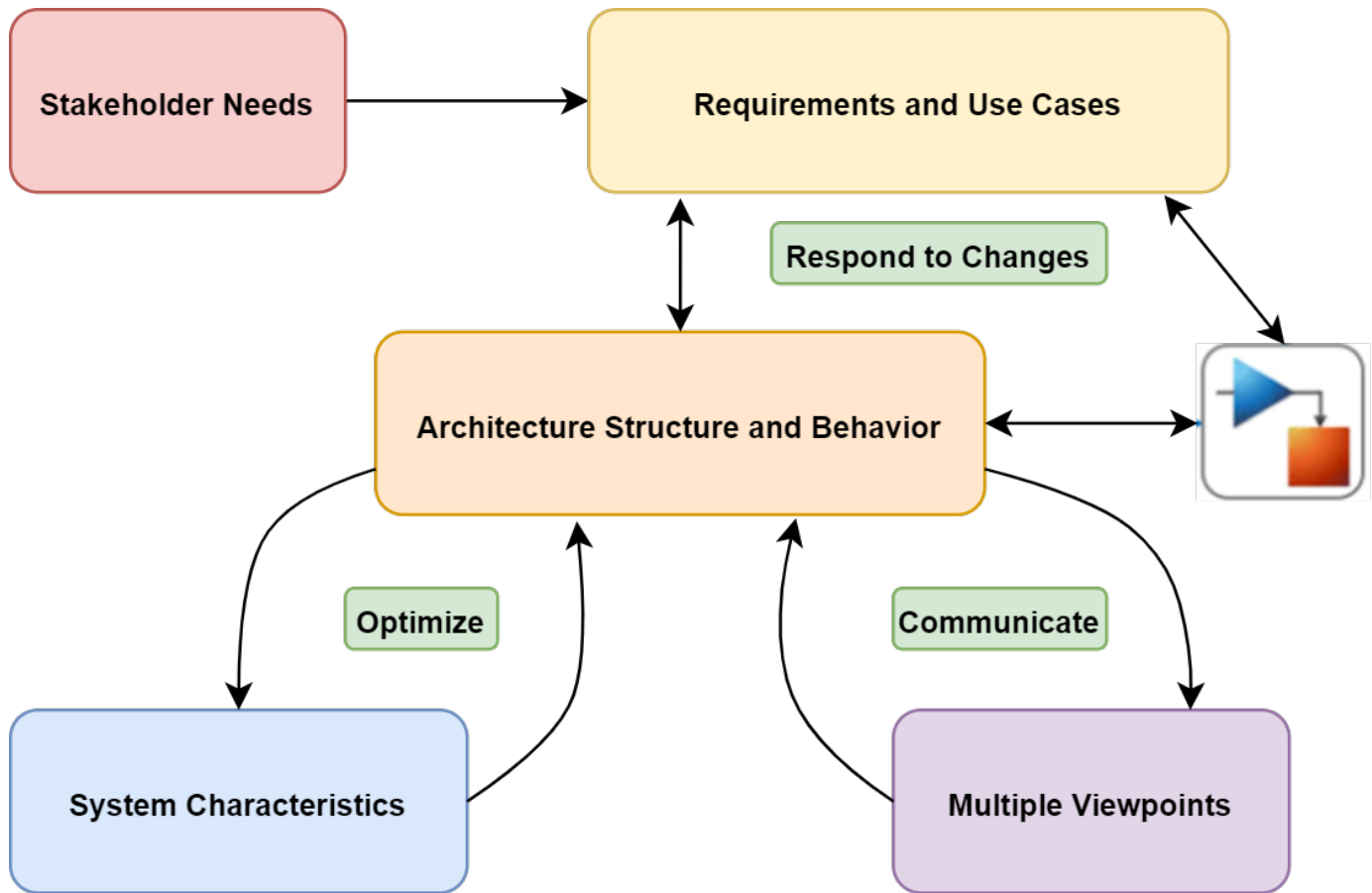
A system refers to a composition of elements that interact to achieve a goal no single element could accomplish on its own. The constituent elements of a system can include mechanical parts, electrical circuits, computer hardware, and software. A system specification describes the system elements, their characteristics and properties, their interactions with each other, and the desired interaction (or interface) of the overall system with its environment.

System Composer allows you to describe systems in terms of architecture models as a combination of structural elements with underlying behavioral descriptions. These descriptive models can sometimes be presented as distinct diagrams that are consistent with each other.

To perform a basic systems engineering workflow to design a mobile robotic arm using System Composer, follow these steps.

- “Create Architecture Model with Interfaces and Requirement Links” on page 2-5
- “Extend Architectural Design Using Stereotypes” on page 2-19
- “Analyze Architecture Model with Analysis Function” on page 2-25
- “Inspect Components in Custom Architecture Views” on page 2-30
- “Implement Behaviors for Architecture Model Simulation” on page 2-37

The model-based systems engineering (MBSE) workflow enabled by System Composer involves starting with stakeholder needs, identifying requirements and use cases, designing an architecture iteratively, and implementing functionality using behavior models. You can also use analyses and trade studies to optimize architectural design and communicate facets of the system using architecture views. This figure illustrates an MBSE workflow.



With System Composer, you can implement a systems engineering workflow.

1 Author architecture models and define system requirements:

- Create hierarchical models of system structure that represent functional, logical, or physical decompositions of the system using components, ports, and connectors.
- Import models from MATLAB® tables and export them with System Composer changes.
- Edit and view the instance-specific parameters specified as model arguments on a component or architecture using the **Parameter Editor**.
- Create and manage data interfaces between structural architectural elements using the **Interface Editor**.
- Manage model-to-model allocations to show relationships between software components and hardware components and to indicate deployment strategies using the **Allocation Editor**.
- Refine and elaborate requirements using Requirements Toolbox™ in the **Requirements Editor**. Link requirements to architectural model elements.

2 Define metadata, generate views, describe system behavior, and analyze architectures:

- Extend base architectural elements to create domain-specific conceptual representations using the **Profile Editor**.
- Filter views of the system structure using a component diagram, hierarchy diagram, or class diagram in the **Architecture Views Gallery**.

- Represent the interaction between structural elements of an architecture as a sequence of message exchanges with a sequence diagram in the **Architecture Views Gallery**.
 - Perform static analysis and trade studies to optimize architectures using the **Instantiate Architecture Model** and the **Analysis Viewer** tools.
- 3** Implement component behavior and use simulation-based workflows to verify requirements:
- Specify component behavior using block diagrams in Simulink, state machines in Stateflow[®], and physical interfaces in Simscape[™] using subsystem behaviors.
 - Design a software architecture model, define the execution order of the functions from the components in the **Functions Editor**, simulate the design at the architecture level, and generate code.
 - Verify and validate requirements with Simulink Test[™] using the **Test Manager**.
 - Generate reports using Simulink Report Generator[™]. For more information, see “System Composer Report Generation for System Architectures”.

For definitions and applications of common System Composer terms and concepts, see “System Composer Concepts” on page 3-2.

Create Architecture Model with Interfaces and Requirement Links

In this section...

“Visually Represent System” on page 2-5

“Edit Data Interfaces” on page 2-13

“Decompose Components” on page 2-15

“Robot Arm Architecture Model” on page 2-16

“Manage Requirement Links” on page 2-17

Create an architecture model of a robot arm using System Composer. Define interfaces on ports and link requirements on components. When you complete these steps, you will have created a completed Robot model.

A Requirements Toolbox license is required to link, trace, and manage requirements in System Composer.

For more information about the model-based systems engineering workflow within System Composer, see “Compose and Analyze Systems Using Architecture Models” on page 2-2.

Visually Represent System

Implementing an architectural design starts with visually representing the system using components and their connections. Create an architecture model, represent the system components, and draw the connections between them.

Create Architecture Model

- 1 Enter this command in the MATLAB Command Window.

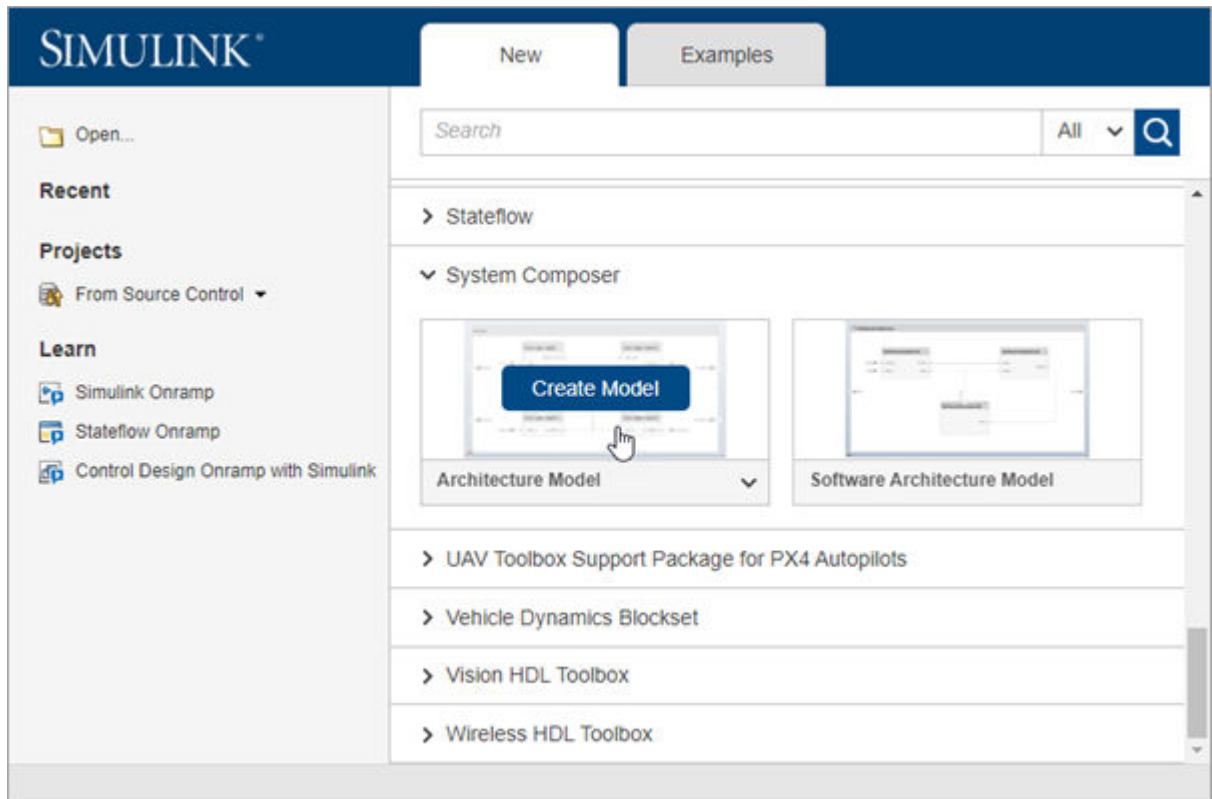
```
systemcomposer
```

The Simulink Start Page opens to System Composer.

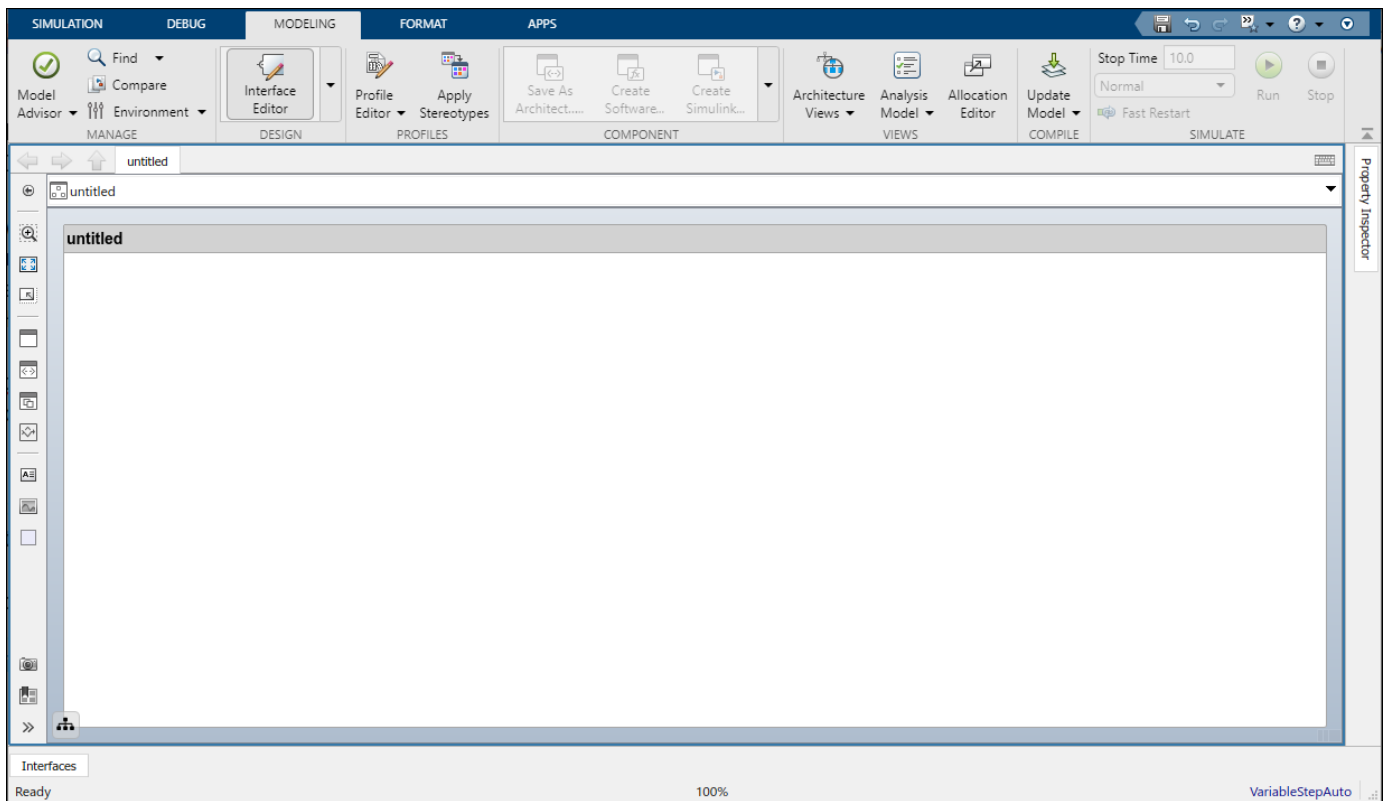
- 2 Click **Architecture Model**.

Use a System Composer **Architecture Model** to describe systems as a combination of structural elements with underlying behavioral descriptions. Use a **Software Architecture Model** to easily define the execution order of your functions from your components, simulate your design in the architecture level, and generate code by linking your Simulink export-function, rate-based, or JMAAB models to components.

For more information about software architecture models, see “Author Software Architectures”.

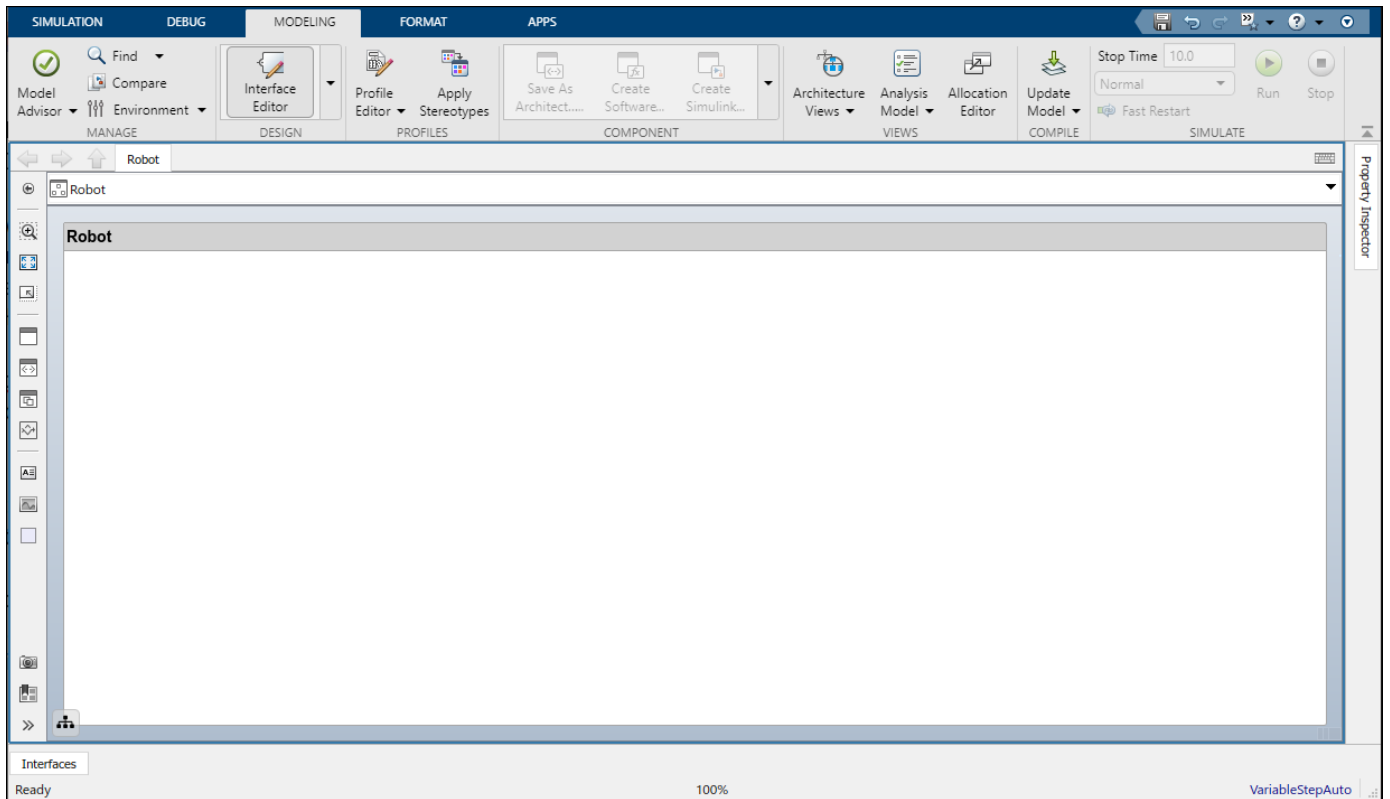


A new, blank architecture model canvas opens. You can identify an architecture model by the badge in the lower left corner and the component palette on the left side.



- 3 Double-click the architecture model header and change `untitled` to a descriptive model name, for example, `RobotDesign`. The name of the model generally reflects the system whose architecture you are building.

2 Compose an Architecture Model

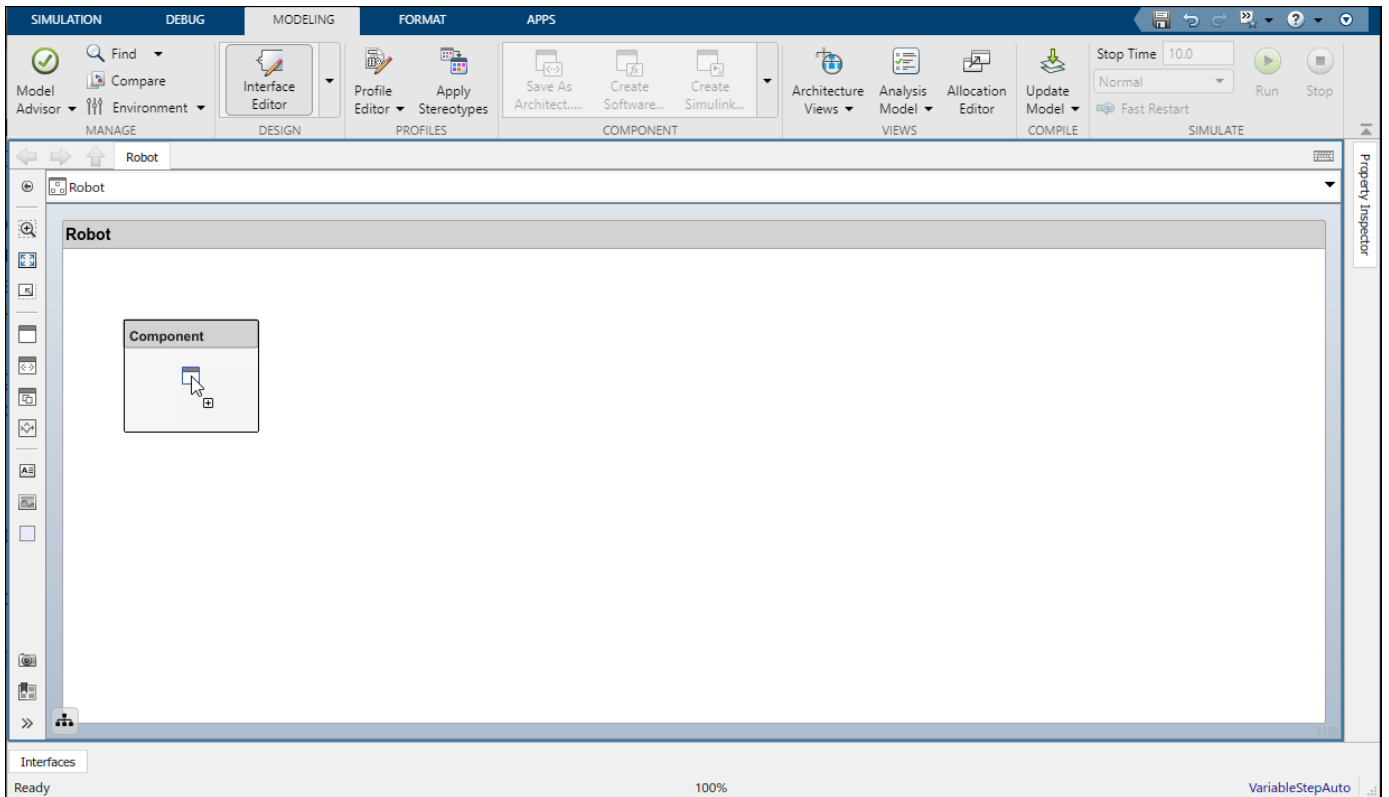


- 4 Save the model.

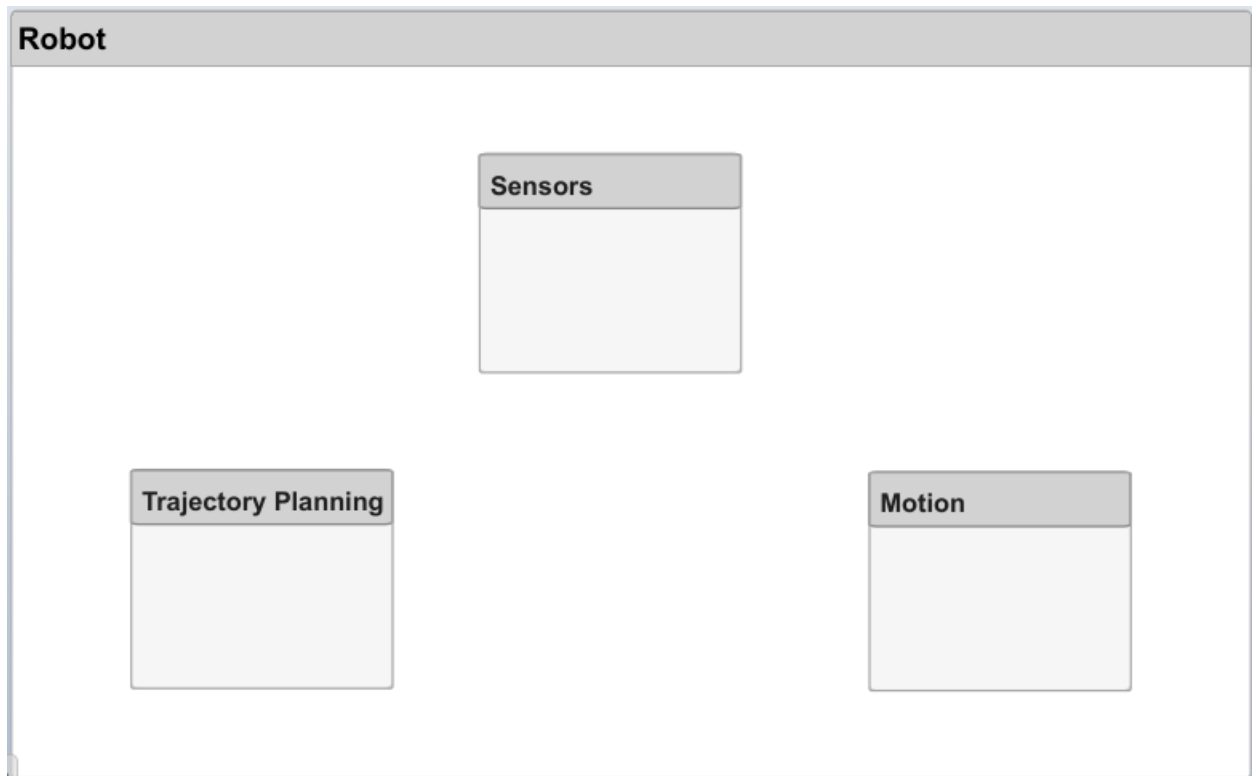
Draw Components

Design a mobile robotic arm where a sensor senses position and trajectory planning computes a path to a location that the robot needs to reach using motion. An architecture model of such a system could consist of three primary components: **Sensors**, **Trajectory Planning**, and **Motion**. You can represent these components in System Composer using three Component blocks.

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



- 2 Rename the component as Sensors.
- 3 Follow these steps to create Trajectory Planning and Motion components.

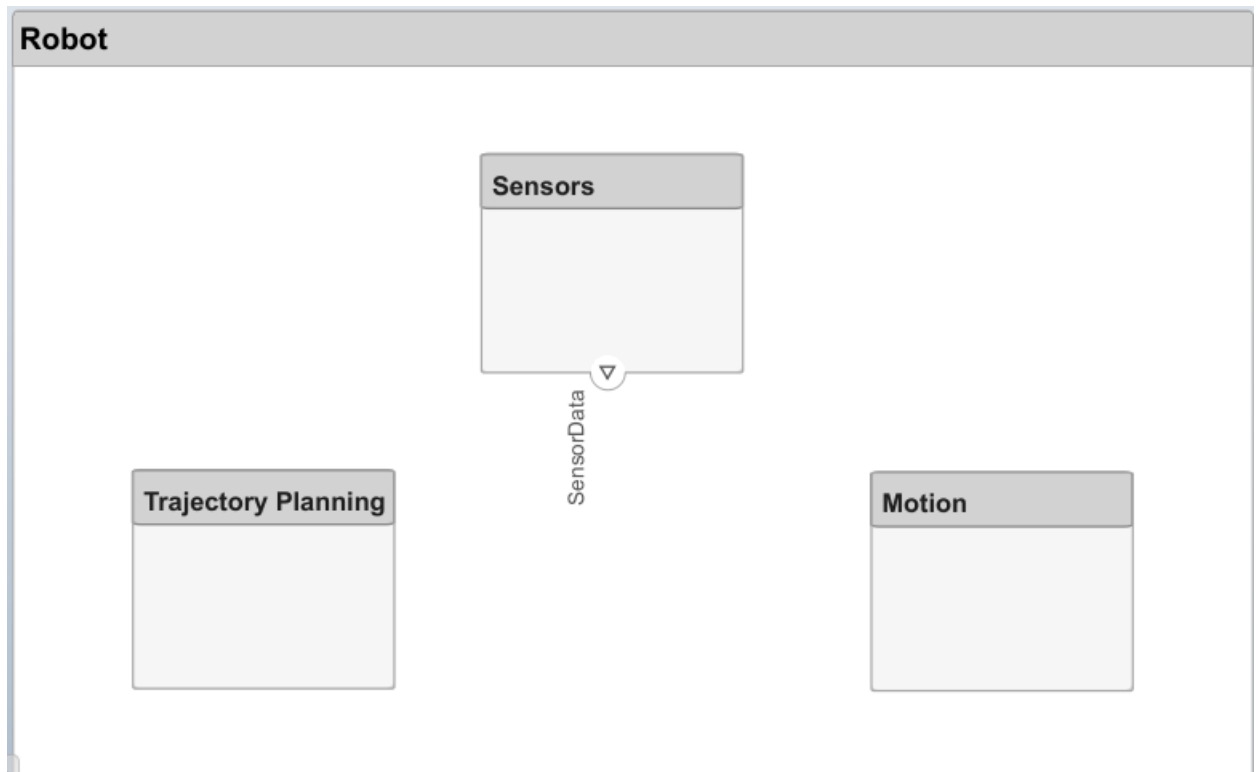



Create Ports and Connections

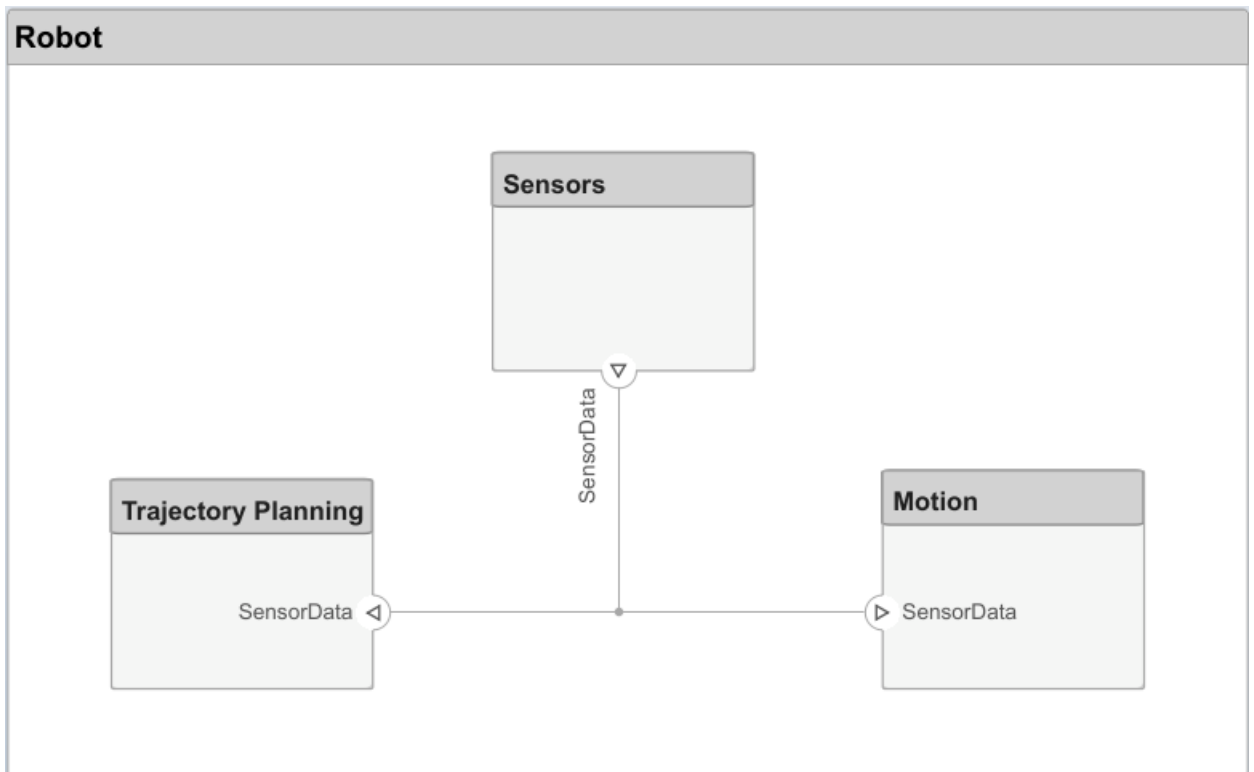
You can begin to create connectivity between components by describing the flow of power, energy, data, or any other representative information. Create ports on the components that provide or consume information and connectors that bind two component ports to represent the flow of the information.

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 your cursor over a component side. Click and release to view port options. Select either **Input**, **Output**, or **Physical** to create a port. Rename the port using a name that represents the information that flows through that port.

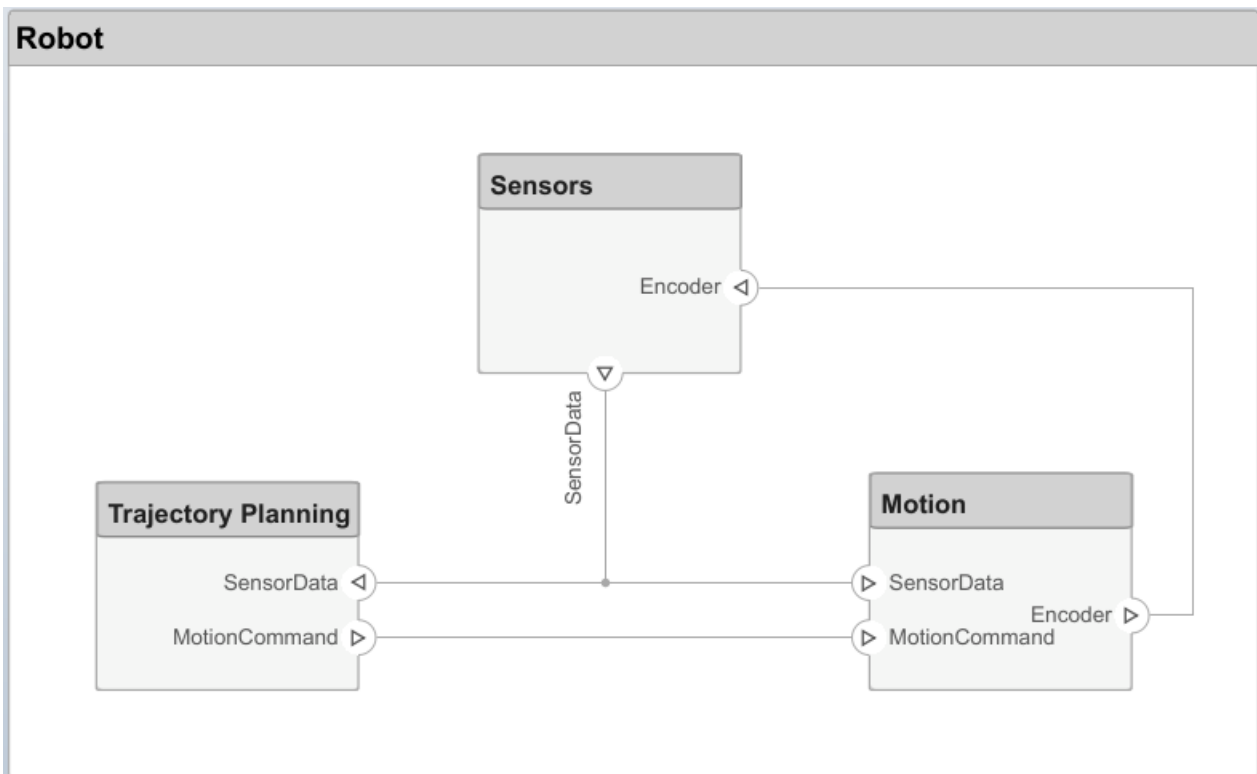
- 1 Create an output port on the bottom side of the **Sensors** component. Rename it **SensorData**.



- 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 pointer. By default, this new port has the same name as the source port.
- 3 Pause 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 the connections as shown in this figure.

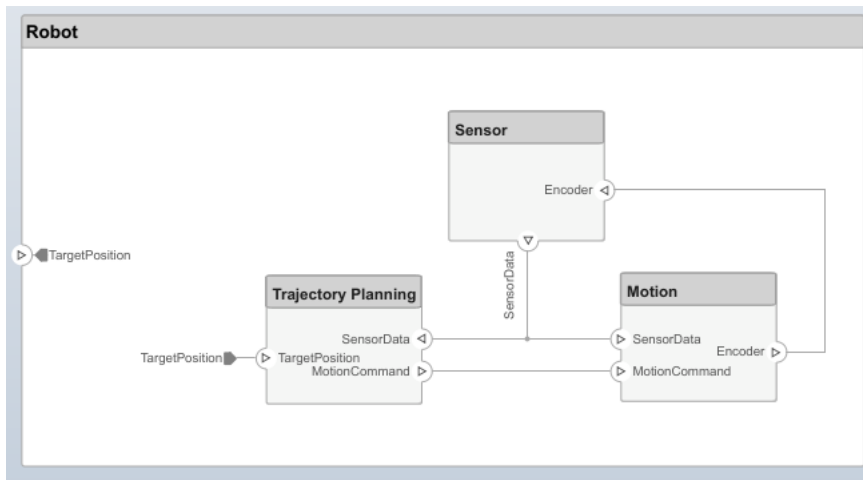


The root level of the architecture model can also have ports that describe the interaction of the system with its environment. In this example, the target position for the robot is provided by a computer external to the robot itself. Represent this relationship with an input port.

- 1 Click the left edge of the architecture model and enter 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 Data Interfaces

You can define a data interface to fully specify a connection and its associated ports. A data interface can consist of multiple data elements with various dimensions, units, and data types. To check for consistency when connecting a port, you can also associate interfaces with unconnected ports during component design.

Specify the information flow through a port between components by configuring the data interface with attributes. A data 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 data 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

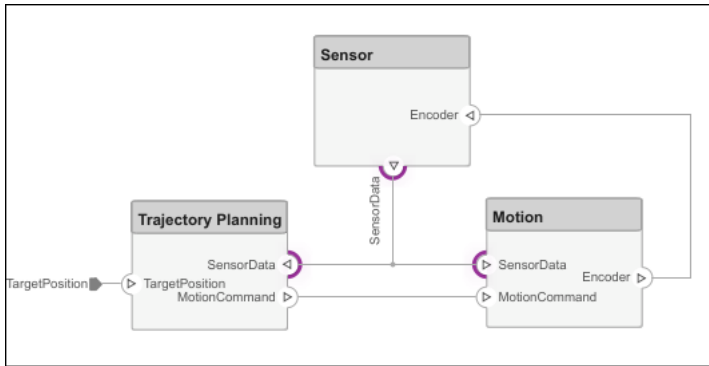
- 1 To open the **Interface Editor**, navigate to **Modeling > Interface Editor**.


- 2 Click the  button to add a data interface. Name the interface `sensordata`.

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

- 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.



- Add a data element to the selected data interface. Click the  button to add a data element and name it `timestamp`.
- Continue adding data elements to the data interface as specified by clicking the add data element button.

Name	Type	Units
<code>timestamp</code>	<code>double</code>	<code>seconds</code>
<code>position1 for motor 1</code>	<code>double</code>	<code>degrees</code>
<code>position2 for motor 2</code>	<code>double</code>	<code>degrees</code>
<code>distance1 for sensor 1</code>	<code>double</code>	<code>meters</code>
<code>direction1 for sensor 1</code>	<code>double</code>	<code>degrees</code>
<code>distance2 for sensor 2</code>	<code>double</code>	<code>meters</code>
<code>direction2 for sensor 2</code>	<code>double</code>	<code>degrees</code>

- Edit the properties of a data element in the **Interface Editor**. Click on the cell corresponding to the data element in the table and add units as shown in the specification.

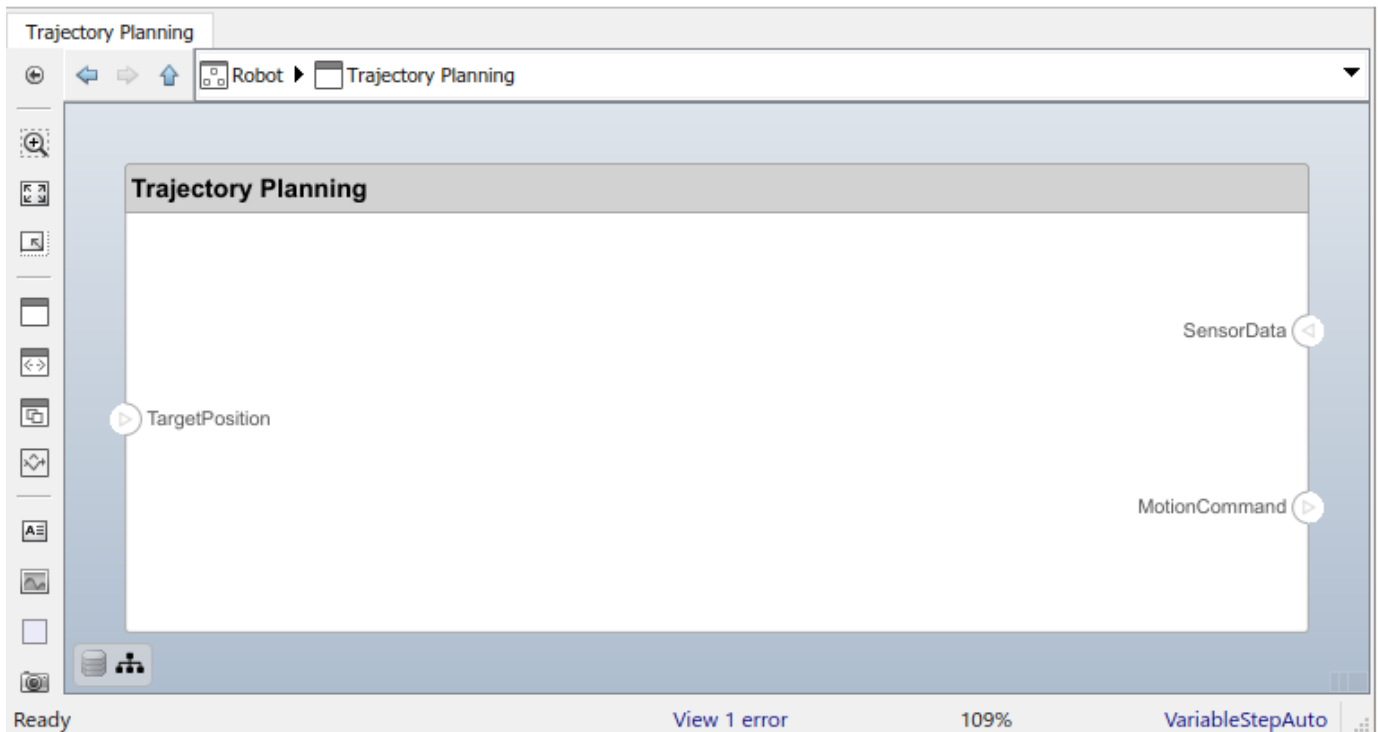
Click the drop-down list next to the  button to save the data interface to a data dictionary. A data dictionary allows you to collectively manage and share a set of interfaces among models. For instance, later in the design, if you choose to model the external computer as a separate architecture model,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

Interfaces		
	Type	Units
RobotDD.sidd		
sensordata		
timestamp	double	seconds
direction1	double	degrees
direction2	double	degrees
distance1	double	meters
distance2	double	meters
position1	double	degrees
position2	double	degrees

Decompose Components

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

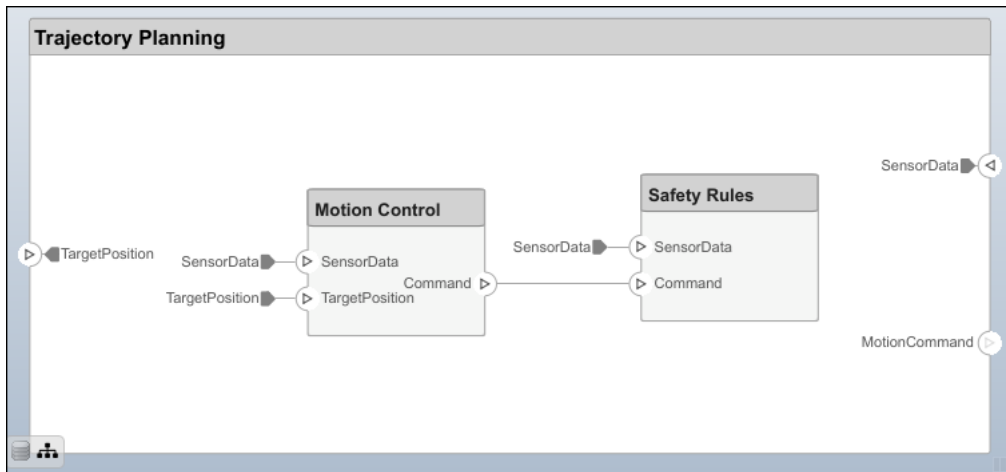
- 1 Double-click the Trajectory Planning component. The Explorer Bar and **Model Browser** indicates the position of the component in the model hierarchy.



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.

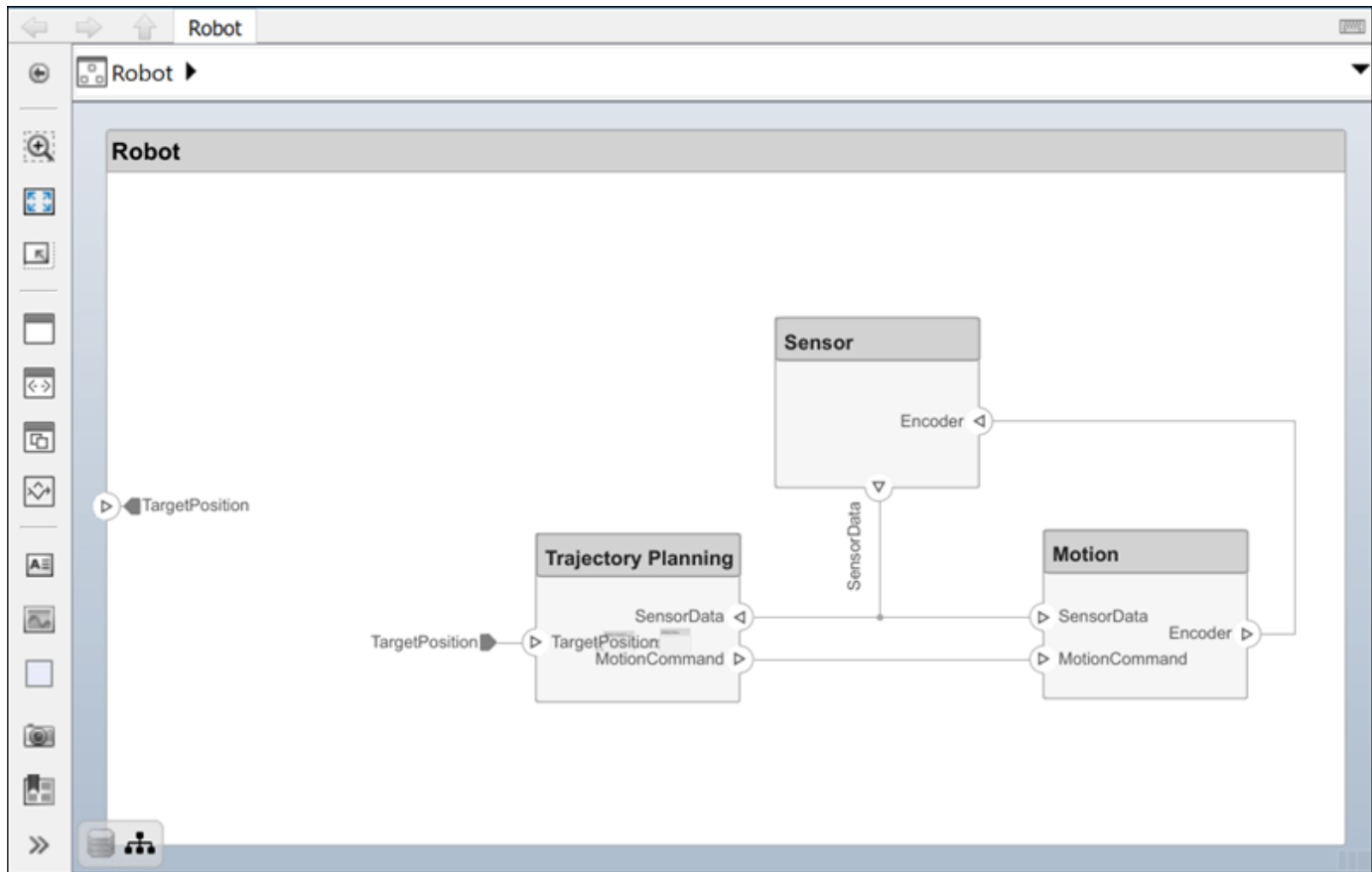
- 2 Add `Motion Control` and `Safety Rules` components as part of the `Trajectory Planning` architecture.

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



Robot Arm Architecture Model

Open the architecture model of a robot arm that consists of sensors, motion actuators, and a planning algorithm. You can use System Composer to view the interfaces and manage the requirements for this model.

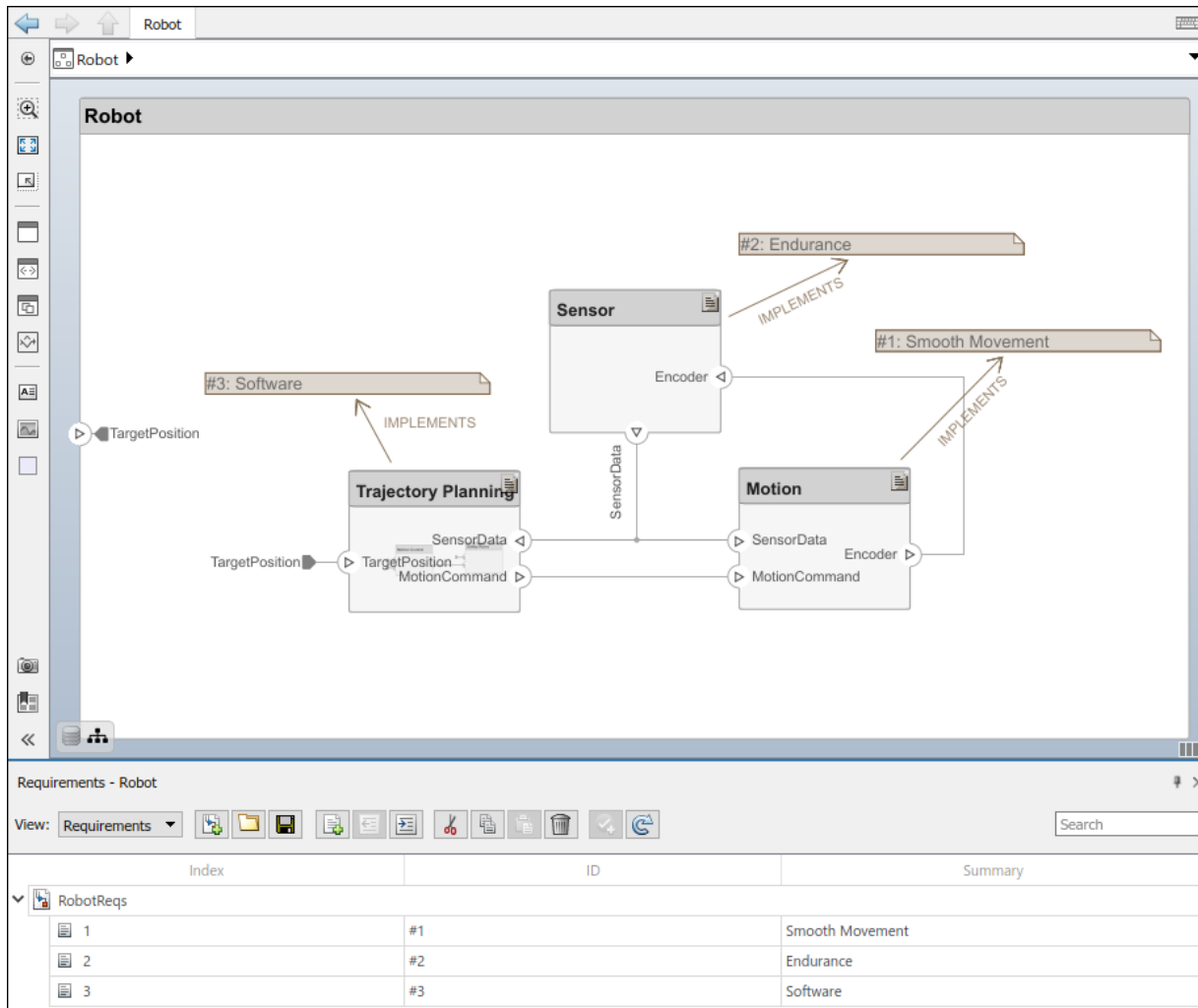


Manage Requirement Links

Requirements are integral to the systems engineering process. Some requirements relate to the functionality of the overall system, and some relate to aspects of performance such as power, size, and weight. Decomposing high-level requirements into low-level requirements and deriving additional requirements is crucial to defining the architecture of the overall system. For instance, the overall power consumption of the robot determines the requirement for the power consumption of the robot controller.

To allocate and trace requirements with system elements, System Composer fully integrates with Requirements Toolbox. To derive appropriate requirements, you must sometimes analyze and specify properties (such as power) for elements of the system including components, ports, or connectors. For example, if the total cost of the system is a concern, a `unitPrice` property is necessary.

Manage requirements from the Requirements Perspective in System Composer using Requirements Toolbox. Navigate to **Apps > Requirements Manager**.



To enhance the traceability of requirements, link requirements to architectural components and ports. When you click a component in the Requirements Perspective, linked requirements are highlighted. Conversely, when you click a requirement, the linked components are shown. To directly create a link, drag a requirement onto a component or a port.

See Also

More About

- “Extend Architectural Design Using Stereotypes” on page 2-19
- “Analyze Architecture Model with Analysis Function” on page 2-25
- “Inspect Components in Custom Architecture Views” on page 2-30
- “Implement Behaviors for Architecture Model Simulation” on page 2-37
- “System Composer Concepts” on page 3-2

Extend Architectural Design Using Stereotypes

In this section...
“Mobile Robot Architecture Model” on page 2-19
“Load Architecture Model Profile” on page 2-20
“Apply Stereotypes to Model Elements” on page 2-22
“Set Properties” on page 2-23

You can add the `unitPrice` property to an electrical component using a stereotype. A stereotype extends the modeling language with domain-specific metadata. A stereotype adds properties to the root-level architecture, component architecture, ports, connectors, data interfaces, value types, functions, requirements, and requirement links. You can also apply a stereotype to only a specific element type, such as component architectures. When a model element has a stereotype applied, 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 allow you to analyze an architecture model.

A profile contains a set of model element stereotypes with custom properties. Each profile contains a set of stereotypes, and each stereotype contains a set of properties. For more information, see “Extend Architectural Elements” on page 3-9.

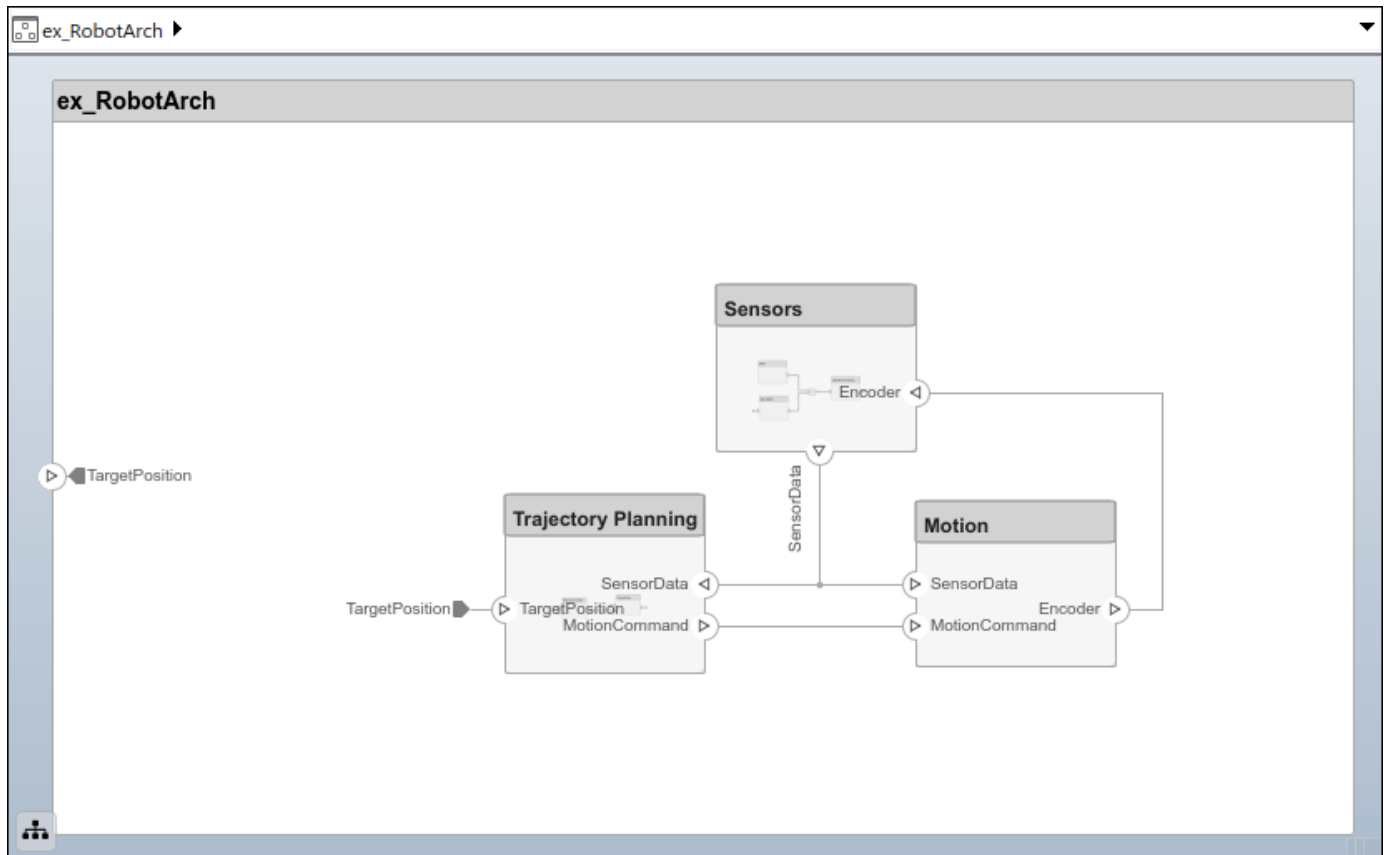
For more information about the model-based systems engineering workflow within System Composer, see “Compose and Analyze Systems Using Architecture Models” on page 2-2.

This example will show you how to compute the total cost of the system given the cost of its constituent parts. The example profile is limited to this goal. Start this tutorial with the following mobile robot architecture model without a profile applied. Use the model to follow the steps and populate its elements with stereotypes and properties.

Mobile Robot Architecture Model

This example shows a mobile robot architecture model with no properties defined. You can apply the stereotypes from the profile `simpleProfile.xml`.

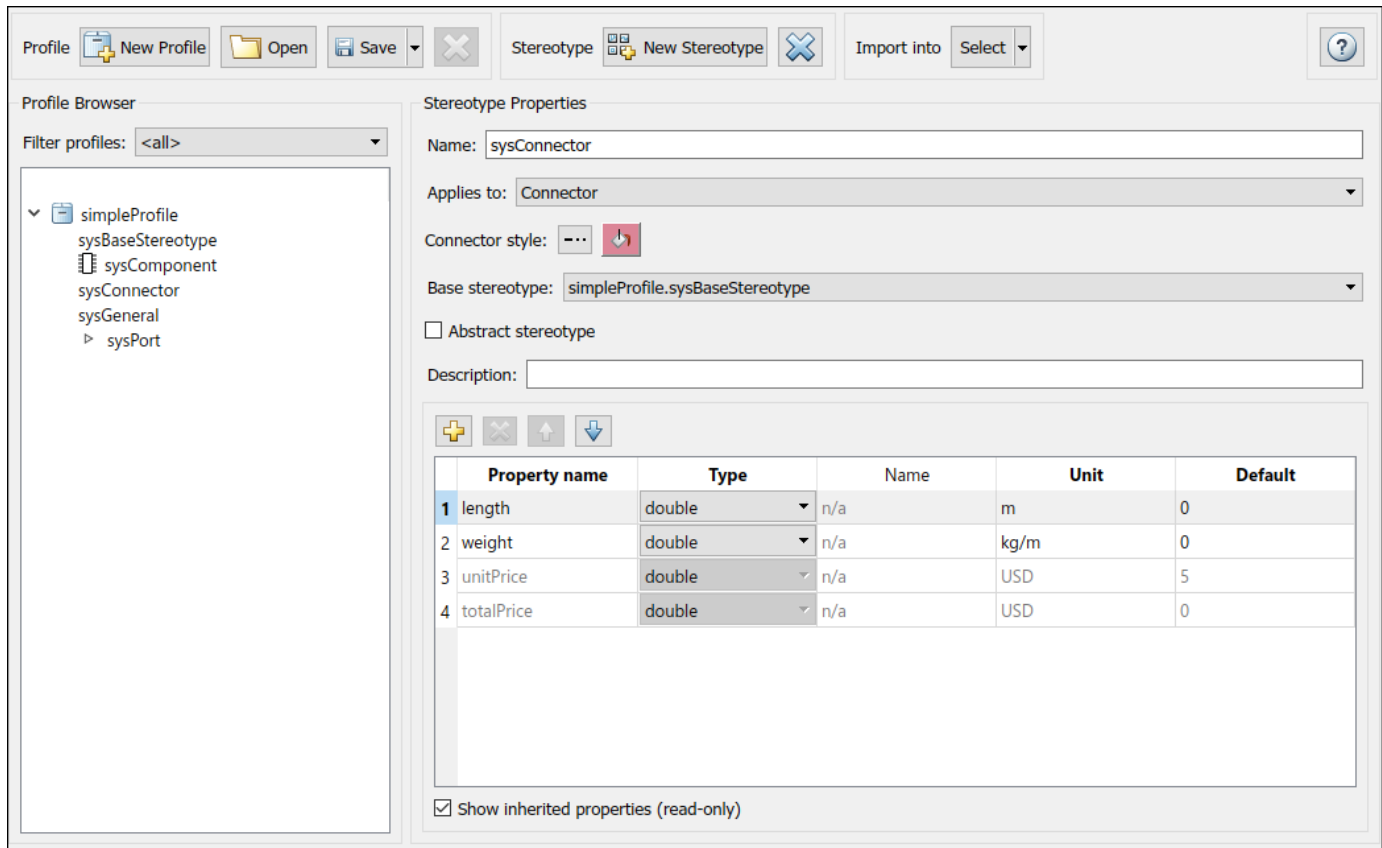
Use the Property Inspector to set the properties on each component.



Load Architecture Model Profile

Load a profile to make stereotypes available for model elements. This procedure uses the model `ex_RobotArch.slx`. Navigate to **Modeling > Profile Editor** to open the **Profile Editor**. Open the profile file `simpleProfile.xml` to load the profile.

In the **Profile Browser**, select the `sysConnector` stereotype. Select **Show inherited properties (read-only)** to view properties inherited from the base stereotype.



In the profile, observe these stereotypes.


Stereotype	Application	Properties
sysBaseStereotype	components, ports, connectors	unitPrice (double, USD, Default: 5) totalPrice (double, USD)
sysComponent	components	weight (double, kg) Inherits properties from sysBaseStereotype
sysConnector	connectors	length (double, m) weight (double, kg/m) Inherits properties from sysBaseStereotype
sysGeneral	components, ports, connectors	ID (int16) Note (string)
sysPort	ports	Inherits properties from sysBaseStereotype

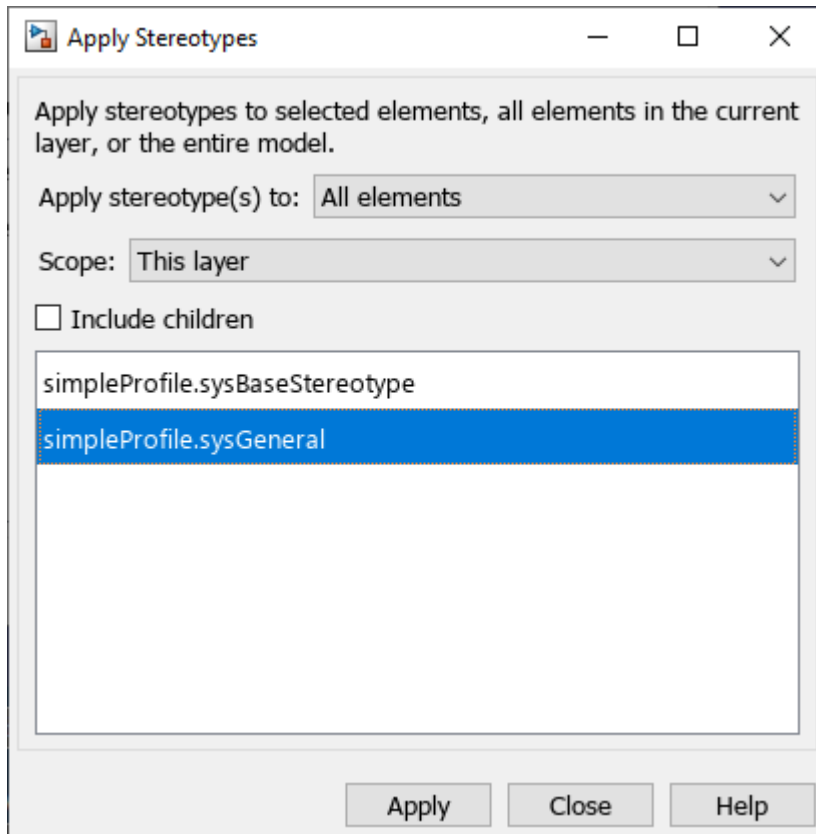
Importing the profile makes stereotypes available to their applicable elements.

- `sysBaseStereotype` stereotype, applicable to all element types, includes shared properties such as `unitPrice` and `totalPrice`.
- `sysComponent` stereotype applies only to components, and includes properties such as `weight` that contributes to the total weight and properties inherited from the `sysBaseStereotype` stereotype with cost specifications of the robot system.
- `sysConnector` stereotype applies to connectors and includes `length` and `weight` properties defined per meter (assuming a physical connector, such as a wire). These properties and the properties inherited from the `sysBaseStereotype` stereotype help compute the total weight and cost of the design.
- `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.
- `sysPort` stereotype applies to ports and does not include any properties except those inherited from `sysBaseStereotype`.

Apply Stereotypes to Model Elements

Add custom properties to a model element by applying a stereotype from a loaded profile.

- 1** On the toolbar, navigate to **Modeling > Profile Editor > Import** .
- 2** Select `simpleProfile`.
- 3** On the toolbar, navigate to **Modeling > Apply Stereotypes** to open the Apply Stereotypes dialog box.
- 4** From **Apply stereotype(s) to**, select `All` elements. From **Scope**, select `This` layer.
In the list of available stereotypes, select `simpleProfile.sysGeneral`.



Click **Apply**.

- 5 From **Apply stereotype(s) to**, select Components. From **Scope**, select Entire model.

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

Click **Apply**.

- 6 From **Apply stereotype(s) to**, select Connectors. From **Scope**, select Entire model.

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

Click **Apply**.

- 7 From **Apply stereotype(s) to**, select Ports. From **Scope**, select Entire model.

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

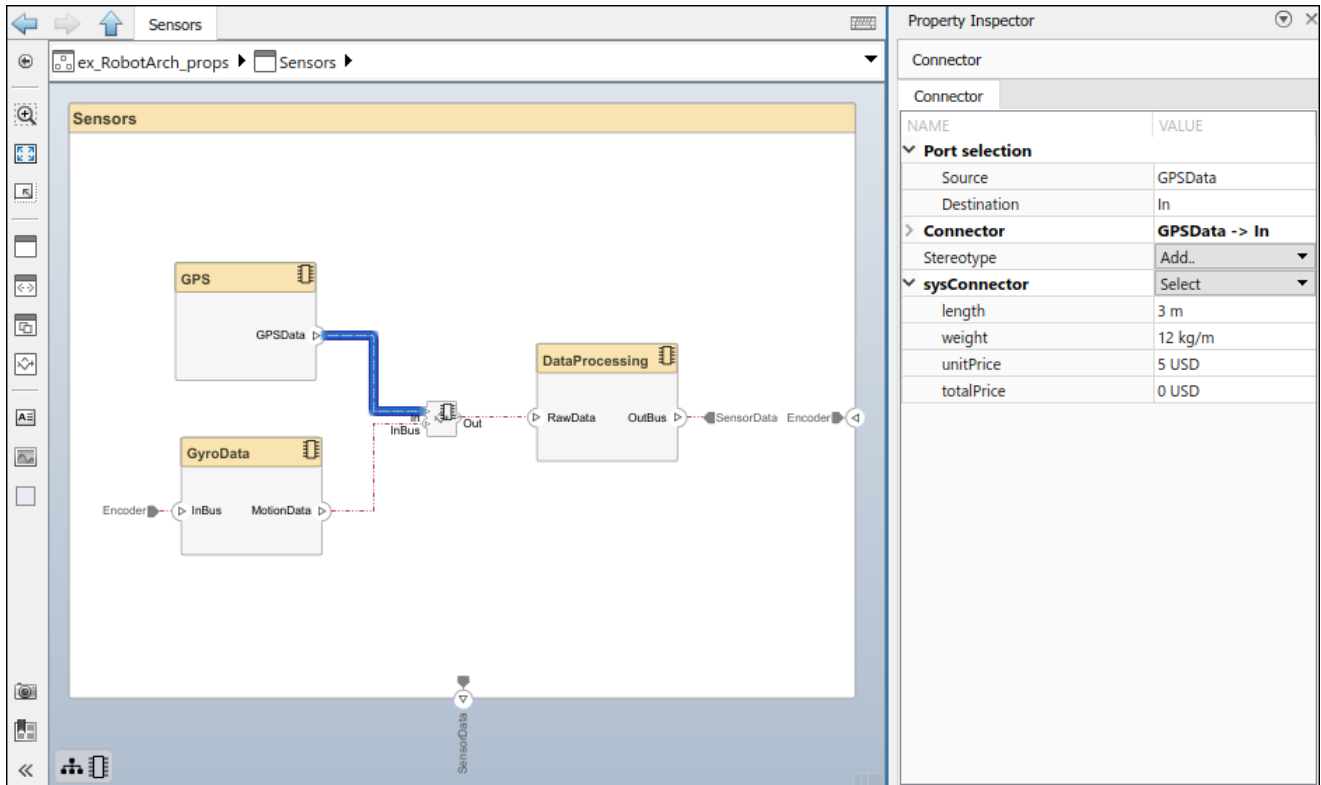
Click **Apply**.

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** by navigating to **Modeling > Property Inspector**.
- 3 Expand the `sysComponent` stereotype to see the properties.

- 4 Set unitPrice to 10 and press **Enter**.
- 5 Select the GPSData port connector. Check that length is set to 3, weight is set to 12, and that unitPrice is set to 5.



- 6 Finish defining metadata across the model for each element using desired property values. Pin the **Property Inspector** to the editor to keep the **Property Inspector** visible during this operation.

Note You can use the `ex_RobotArch_props` architecture model for analysis and view generation because the model includes property values. For more information on analysis, see “Analyze Architecture Model with Analysis Function” on page 2-25. For more information on architecture views, see “Inspect Components in Custom Architecture Views” on page 2-30.

See Also

More About

- “Create Architecture Model with Interfaces and Requirement Links” on page 2-5
- “Analyze Architecture Model with Analysis Function” on page 2-25
- “Inspect Components in Custom Architecture Views” on page 2-30
- “Implement Behaviors for Architecture Model Simulation” on page 2-37
- “System Composer Concepts” on page 3-2

Analyze Architecture Model with Analysis Function

In this section...
“Mobile Robot Architecture Model with Properties” on page 2-25
“Perform Analysis” on page 2-26

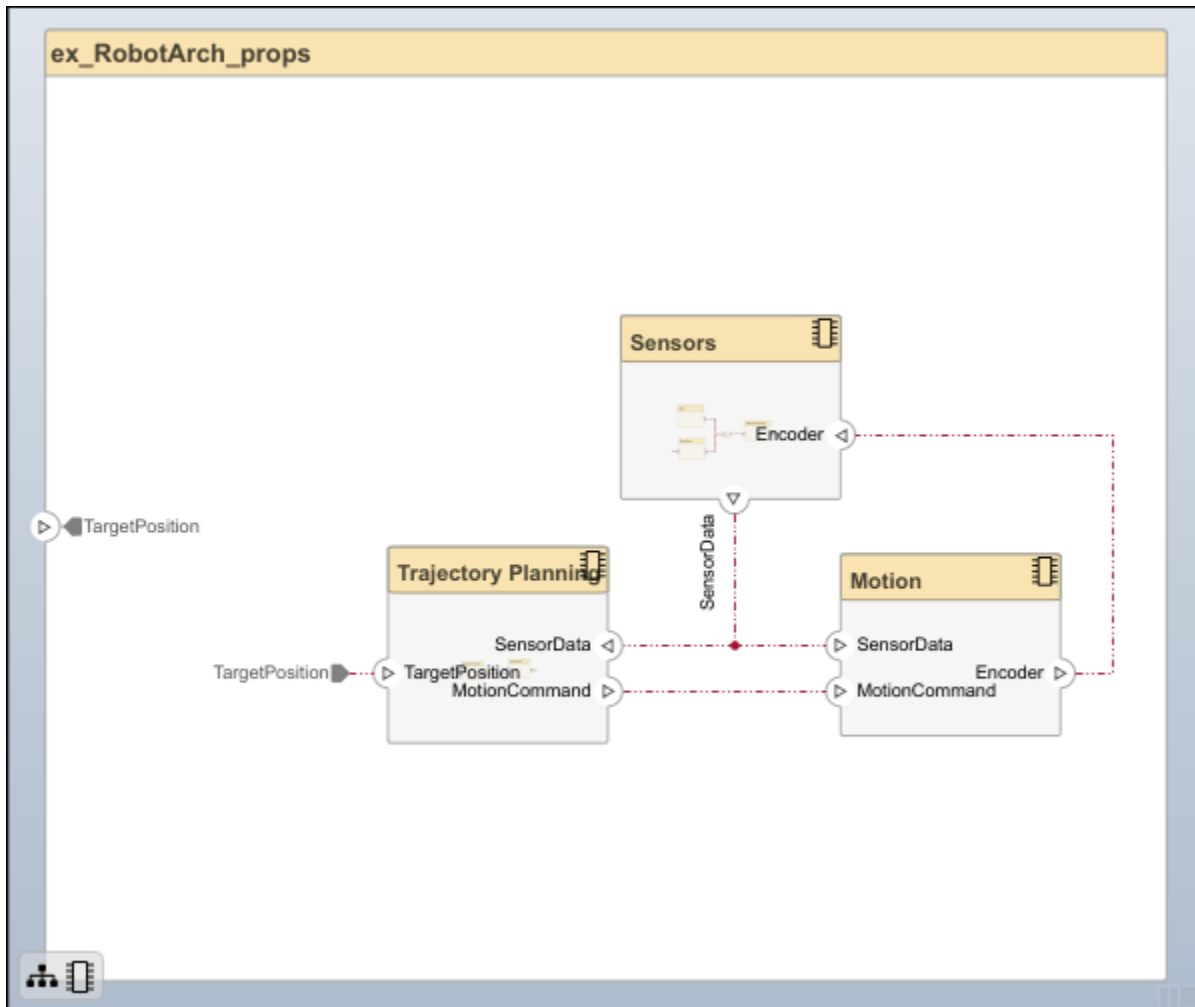
With properties specified on model elements, you can use MATLAB to perform analysis and calculate total cost for all elements within the design. You can then create additional derived requirements for the designers of individual components in the system, such as Trajectory Planning or Sensors.

Perform static analyses based on element properties to perform data-driven trade studies and verify system requirements. Consider a robot architecture model where total cost is a consideration. For this tutorial, you will use the mobile robot architecture model with properties to perform static analysis.

For more information about the model-based systems engineering workflow within System Composer, see “Compose and Analyze Systems Using Architecture Models” on page 2-2.


Mobile Robot Architecture Model with Properties

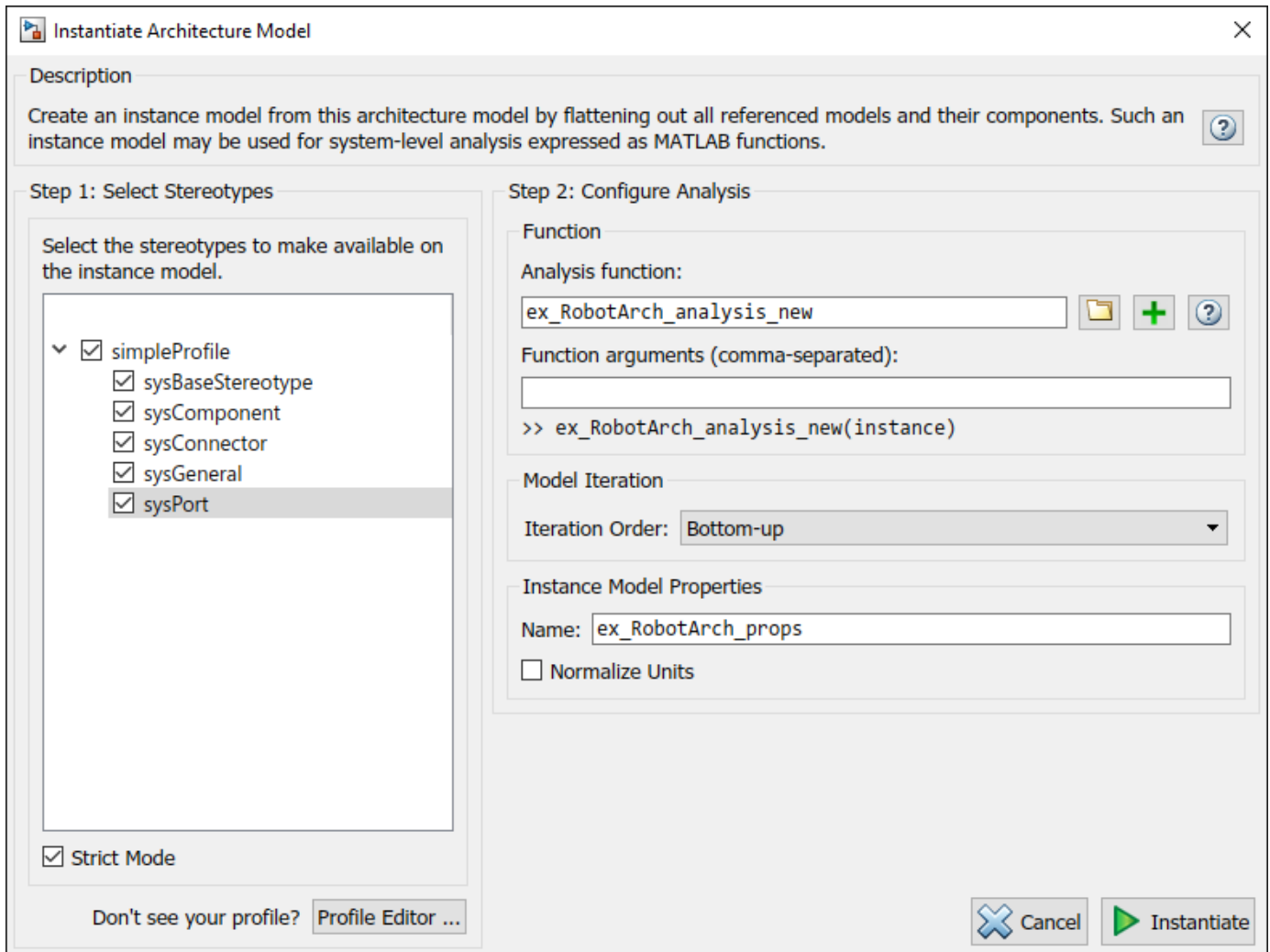
This example shows a mobile robot architecture model with stereotypes applied to components and properties defined.



Perform Analysis

Analyze the total cost for all components in the robot model. This procedure uses the model `ex_RobotArch_props.slx`.

- 1 Navigate to **Modeling > Analysis Model** to open the **Instantiate Architecture Model** tool.
- 2 Add an analysis function. In the **Analysis function** box, enter the function name `ex_RobotArch_analysis_new` without an extension, and then click the  button. A MATLAB function file is created and saved with the name `ex_RobotArch_analysis_new.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 and 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. Copy and paste the function below into your analysis function.

```
function ex_RobotArch_analysis_new(instance,varargin)
    if instance.isComponent()
        if instance.hasValue("sysBaseStereotype.unitPrice")
            sysComponent_totalPrice = instance.getValue("sysBaseStereotype.unitPrice");
        else
            sysComponent_totalPrice = 0;
        end
        if ~isempty(instance.Components)
            for child = instance.Components
                if child.hasValue("sysBaseStereotype.totalPrice")
                    comp_price = child.getValue("sysBaseStereotype.totalPrice");
                    sysComponent_totalPrice = sysComponent_totalPrice + comp_price;
                end
            end
        end
        sysPort_totalPrice = 0;
        for port = instance.Ports
            if port.hasValue("sysBaseStereotype.unitPrice")
                unitPrice = port.getValue("sysBaseStereotype.unitPrice");
            end
        end
    end
end
```

```

        sysPort_totalPrice = sysPort_totalPrice + unitPrice;
    end
end
sysConnector_totalPrice = 0;
for connector = instance.Connectors
    if connector.hasValue("sysBaseStereotype.unitPrice")
        unitPrice = connector.getValue("sysBaseStereotype.unitPrice");
        length = connector.getValue("sysConnector.length");
        sysConnector_totalPrice = sysConnector_totalPrice + unitPrice*length;
    end
end
if (instance.hasValue("sysBaseStereotype.totalPrice"))
    totalPrice = sysComponent_totalPrice + ...
                sysPort_totalPrice + sysConnector_totalPrice;
    instance.setValue("sysBaseStereotype.totalPrice",totalPrice);
end
end
end
end

```

- 3 Return to the **Instantiate Architecture Model** tool, select all the stereotypes, and click **Instantiate**. The **Analysis Viewer** opens and 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, then click **Analyze**.

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

Instances	totalPrice	unitPrice	weight	length	weight	ID
ex_RobotArch_props	814		5	0		
Motion	165	150		7		
Encoder	0	5				
MotionCommand	0	5				
SensorData	0	5				
Sensors	234	78		0		
Adapter	5	5		0		
Adapter.In->Adapter.Out	0	5			0	0
DataProcessing	66	56		5		
OutBus	0	5				
RawData	0	5				
GPS	10	5		47		
GPSData	0	5				
GyroData	15	5		21		
InBus	0	5				
MotionData	0	5				
Adapter.Out->DataProcessing.RawData	0	5			2	12
DataProcessing.OutBus->Sensors.SensorData	0	5			1	12
GPS.GPSData->Adapter.In	0	5			3	12
GyroData.MotionData->Adapter.InBus	0	5			3	12
Sensors.Encoder->GyroData.InBus	0	5			1	12
Encoder	0	5				
SensorData	0	5				
Trajectory Planning	285	45		0		
MotionController	75	60		4		
SensorData	0	5				
TargetPosition	0	5				
command	0	5				
SafetyRules	95	80		4		
OutBus	0	5				
SensorData	0	5				
command	0	5				

The total costs are highlighted in yellow as computed values. The top row represents the grand total for the ex_RobotArch_props architecture.

See Also

More About

- “Create Architecture Model with Interfaces and Requirement Links” on page 2-5
- “Extend Architectural Design Using Stereotypes” on page 2-19
- “Inspect Components in Custom Architecture Views” on page 2-30
- “Implement Behaviors for Architecture Model Simulation” on page 2-37
- “System Composer Concepts” on page 3-2

Inspect Components in Custom Architecture Views

In this section...
“Mobile Robot Architecture Model with Properties” on page 2-30
“Create Spotlight Views from Components” on page 2-31
“Create Filtered Architecture View” on page 2-32

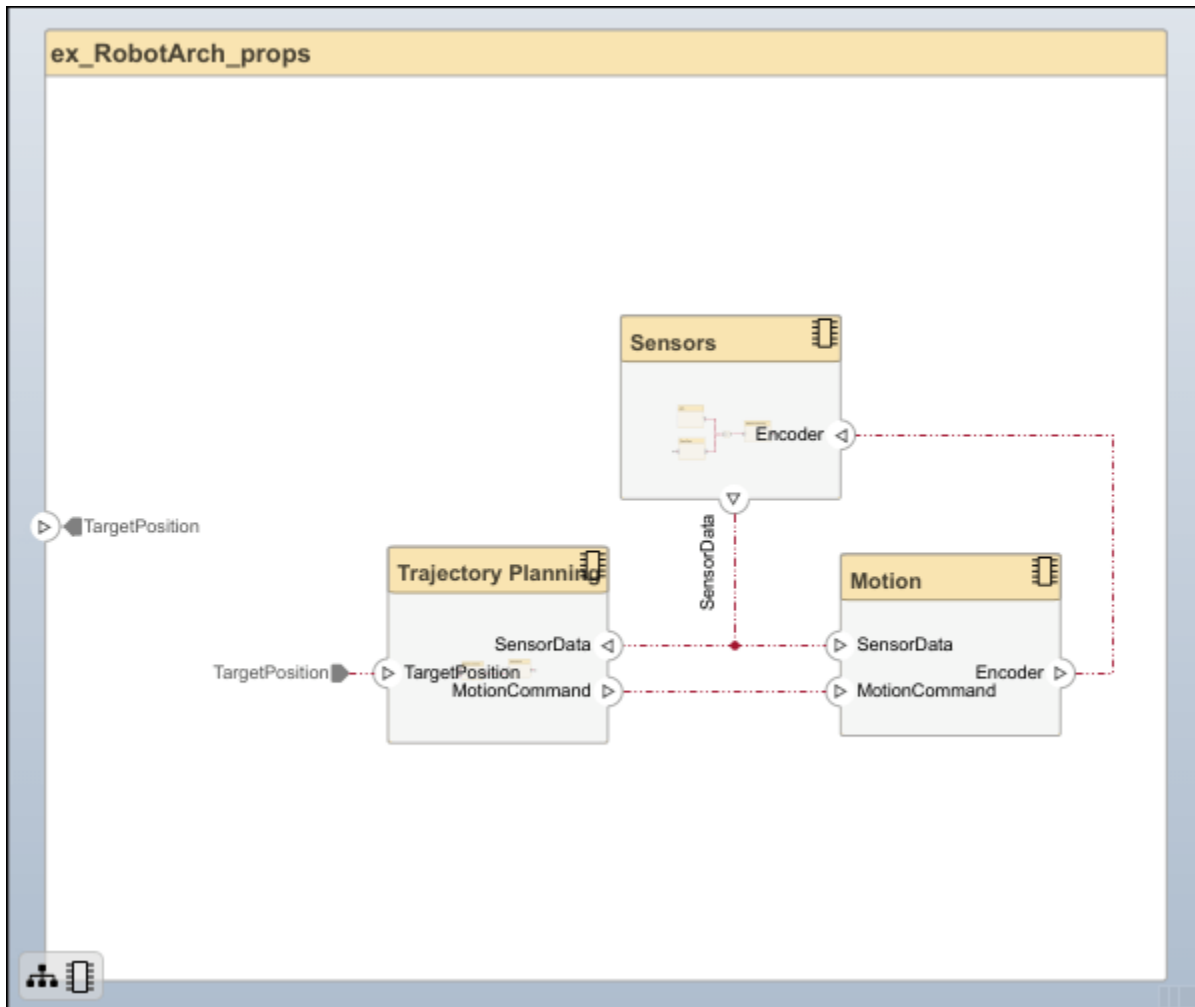
View the hierarchy and connectivity of a component in a specialized view. Specialized views allow you to create simpler diagrams that show only a subset of the original model elements for a specific design activity or concern.

For more information about the model-based systems engineering workflow within System Composer, see “Compose and Analyze Systems Using Architecture Models” on page 2-2.

Use the following System Composer architecture model in this tutorial.

Mobile Robot Architecture Model with Properties

This example shows a mobile robot architecture model with stereotypes applied to components and properties defined.



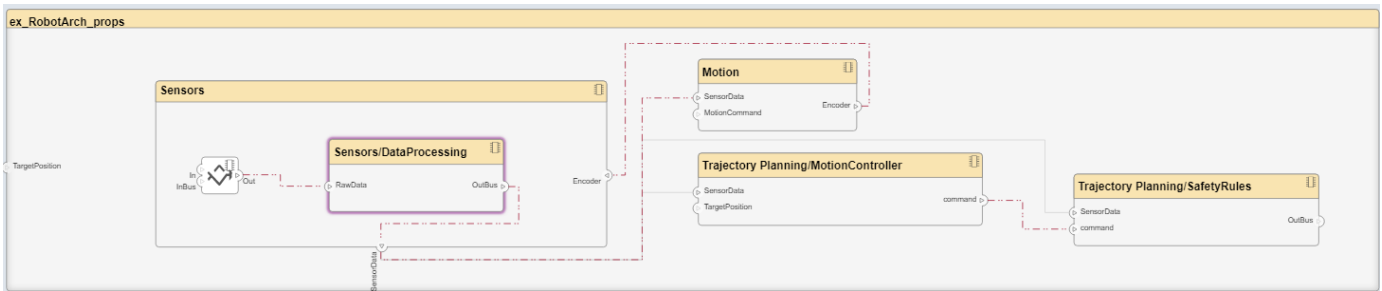
Create Spotlight Views from Components


Create views dynamically using spotlight views.

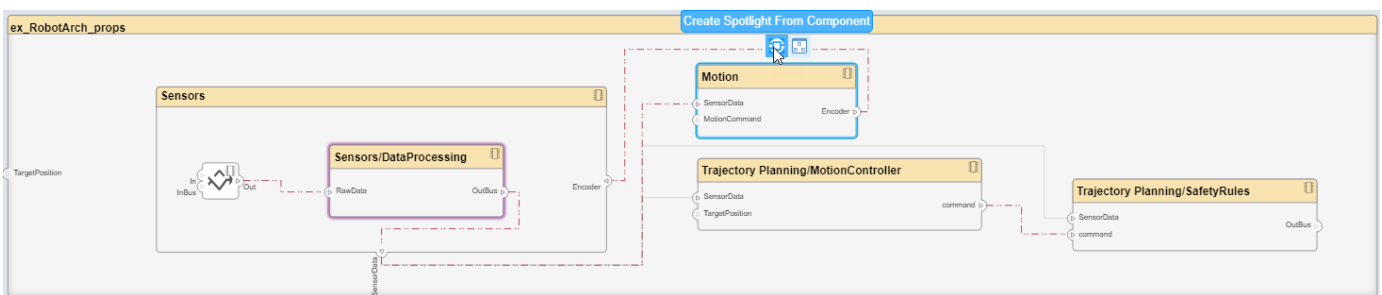
- 1 Double-click the `Sensors` component, then select the `DataProcessing` component.
- 2 Select the `DataProcessing` component and navigate to **Modeling > Architecture Views > Spotlight**. Alternatively, right-click the `DataProcessing` component and select **Create Spotlight from Component**.


The spotlight view launches and shows all model elements to which the `DataProcessing` component connects. The spotlight diagram is laid out automatically and cannot be edited. However, it allows you to inspect just a single component and study its connectivity to other components.

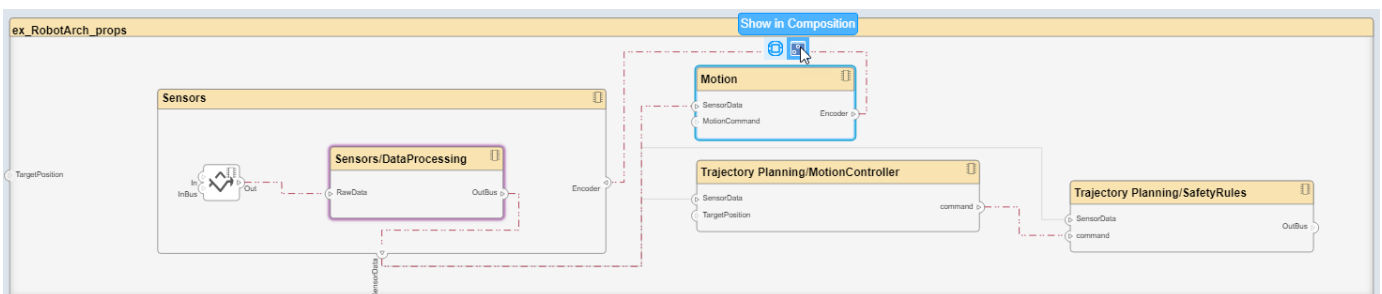
Note Spotlight views are transient. They are not saved with the model.



- 3 Shift the spotlight to another component. Select the **Motion** component. Click the ellipsis above the component to open the action menu. To create a spotlight from the component, click .



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




- 4 To return to the architecture model view, click .

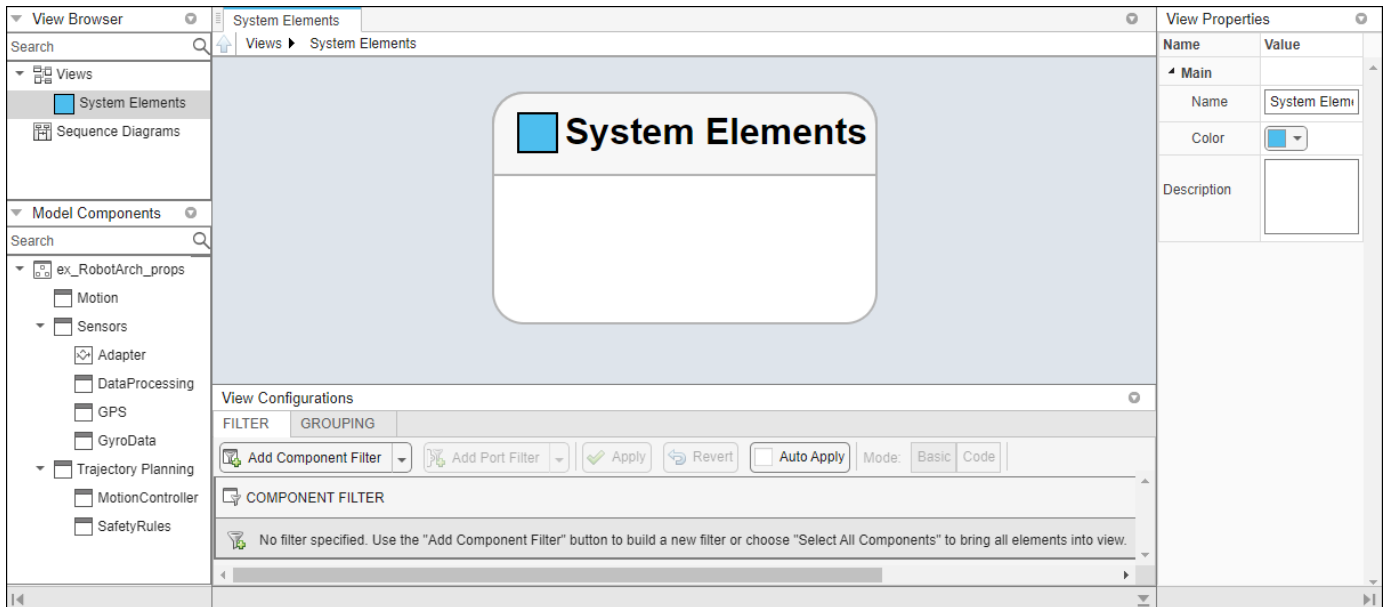
You can make the hierarchy and connectivity of a component visible at all times during model development by opening the spotlight view in a separate window. To show the spotlight view in a dedicated window, in the component context menu, select **Open in New Window**, then create the spotlight view. Spotlight views are dynamic and transient: any change in the composition refreshes any open spotlight views, and spotlight views are not saved with the model.

Create Filtered Architecture View

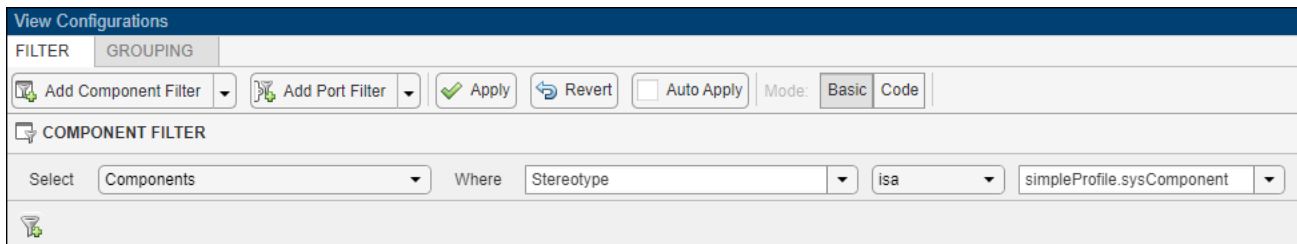
Create filtered architecture views to demonstrate specific perspectives with a component diagram or a hierarchy diagram.

- 1 Navigate to **Modeling > Architecture Views** to open the **Architecture Views Gallery**.

- 2 Select  **New > View** to create a new view.
- 3 In **View Properties** on the right pane, in the **Name** box, enter a name for this view, for example, **System Elements**. If necessary, choose a **Color** and enter a **Description**.



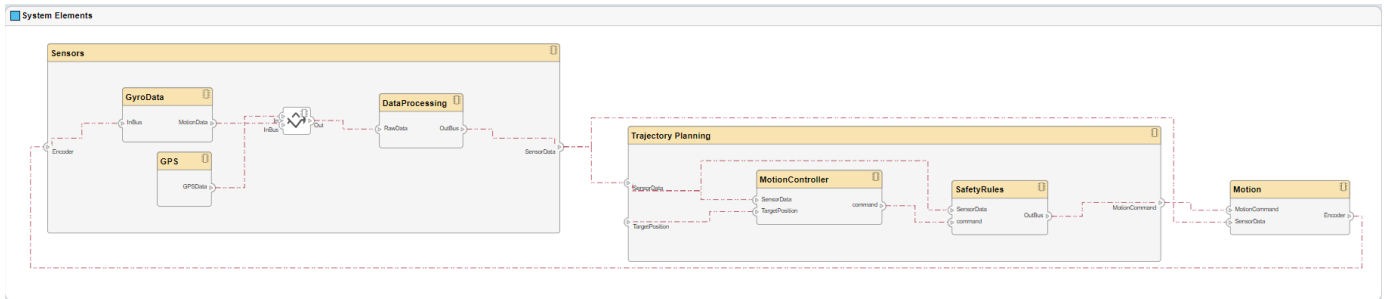
- 4 In the bottom pane on **View Configurations**, from the **Filter** tab, click **Add Component Filter** to add new form-based criterion to a component filter.
- 5 From the **Select** list, select **Components**. From the **Where** list, select **Stereotype**. In the text box, select **simpleProfile.sysComponent** from the list.



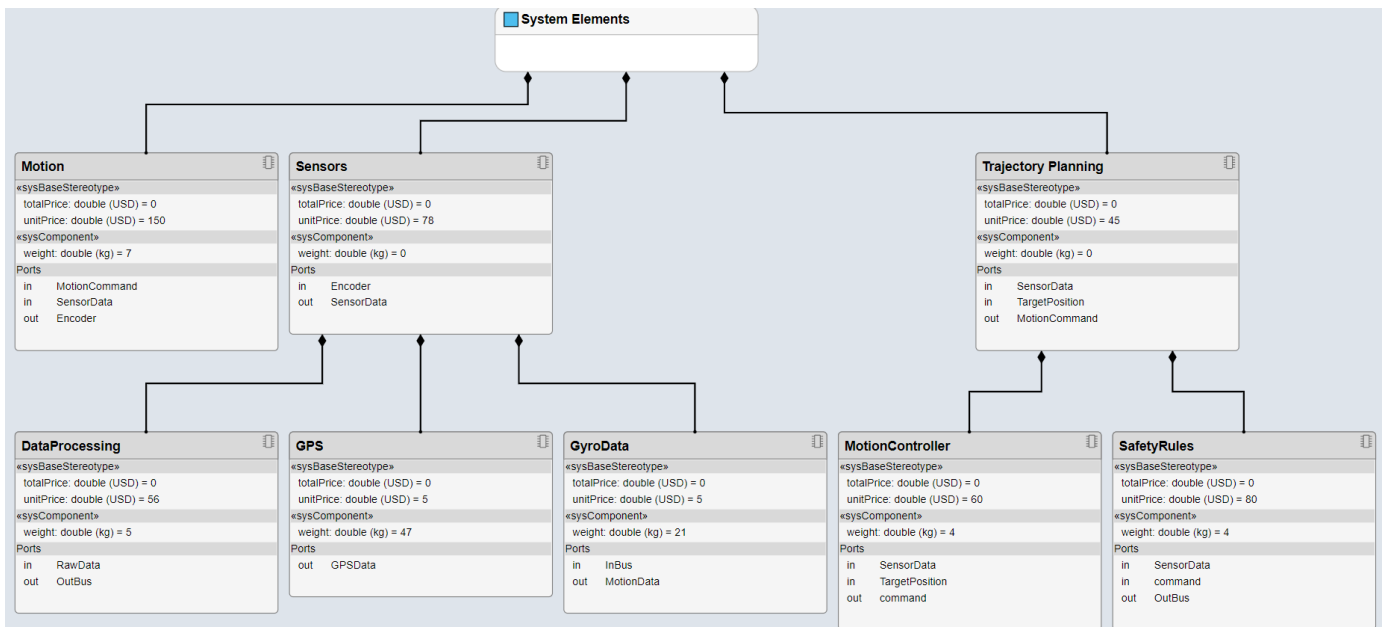
- 6 Click **Apply** .

An architecture view is created using the query in the **Component Filter** box. The view is filtered to select all components with the `simpleProfile.sysComponent` stereotype applied to them.

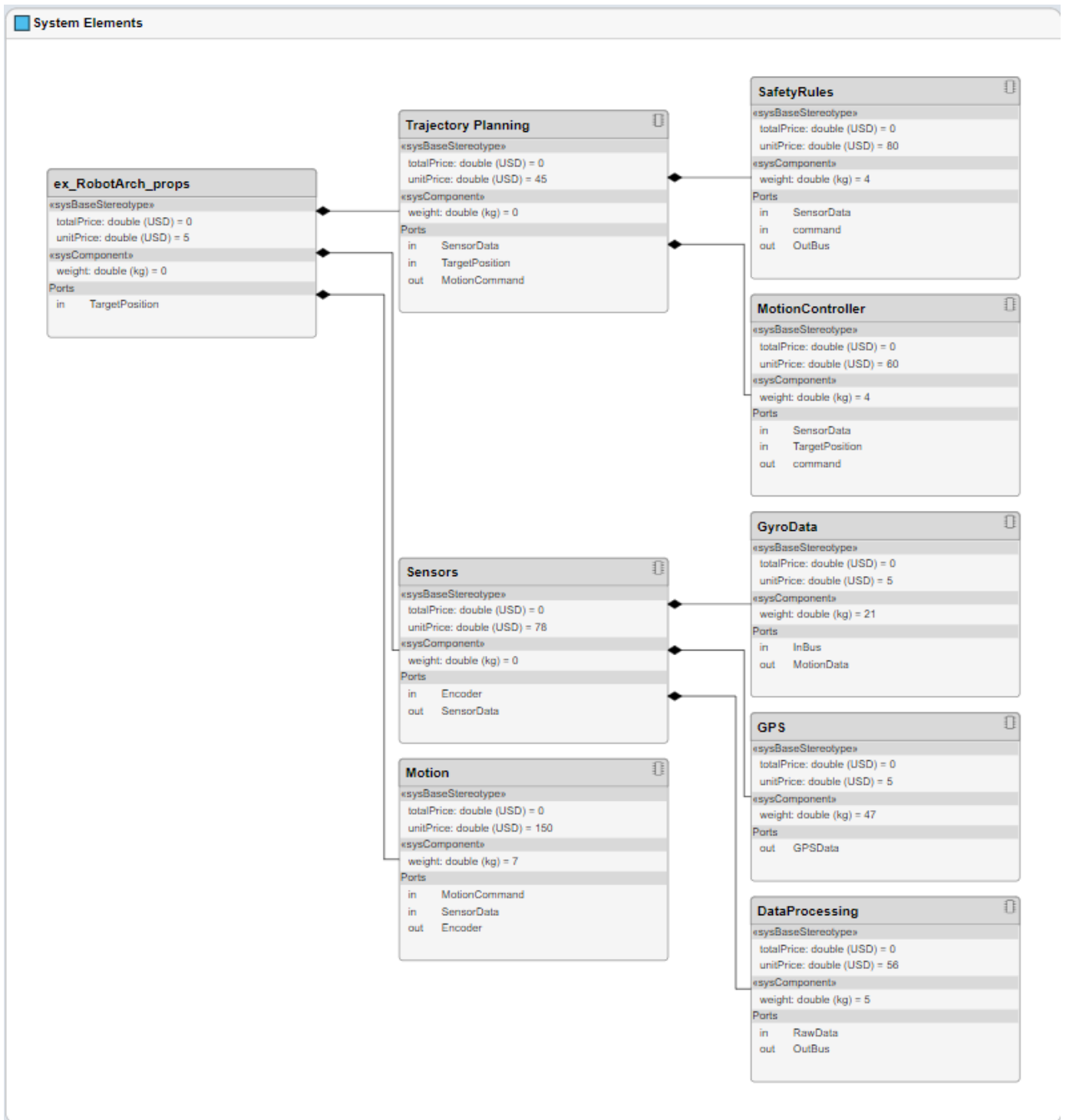
2 Compose an Architecture Model



- 7 In the **Diagram** section of the toolstrip, click **Component Hierarchy** to display the components in tree form with parents above children.



- 8 In the **Diagram** section of the toolstrip, click **Architecture Hierarchy** to display unique architecture types and their relationships using composition connections.



See Also

More About

- “Create Architecture Model with Interfaces and Requirement Links” on page 2-5
- “Extend Architectural Design Using Stereotypes” on page 2-19
- “Analyze Architecture Model with Analysis Function” on page 2-25
- “Implement Behaviors for Architecture Model Simulation” on page 2-37
- “System Composer Concepts” on page 3-2

Implement Behaviors for Architecture Model Simulation

In this section...
“Robot Arm Architecture Model” on page 2-37
“Reference Simulink Behavior Model in Component” on page 2-38
“Add Stateflow Chart Behavior to Component” on page 2-41
“Design Software Architecture in Component” on page 2-42
“Represent System Interaction Using Sequence Diagrams” on page 2-44

A basic systems engineering workflow in System Composer includes composing an architecture system, defining requirements, adding metadata, performing analyses, and representing the architecture through views. After fulfilling these steps, your system design is closer to meeting stakeholder goals and customer needs.

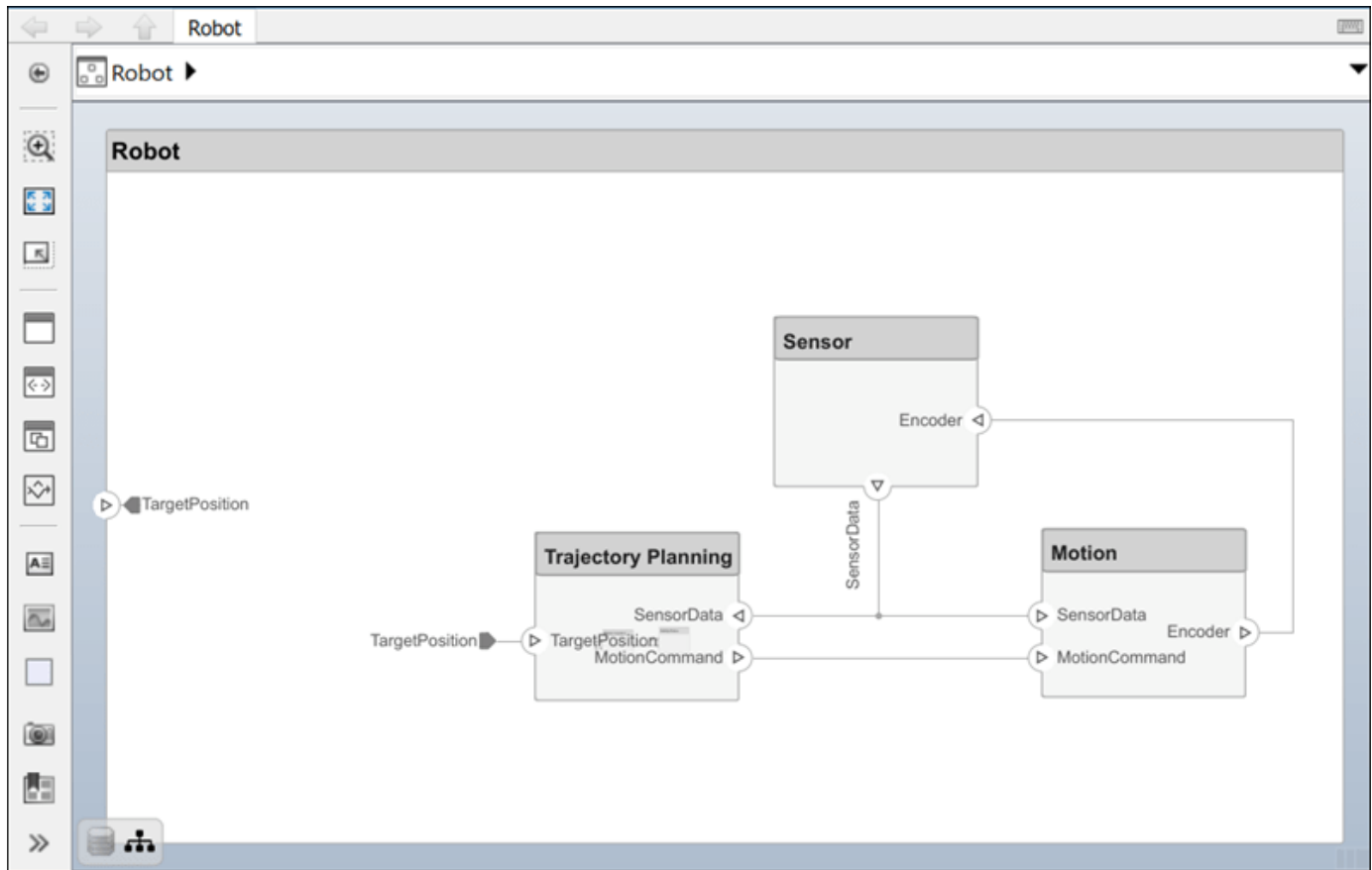
You can also now begin to design the actual system components using Simulink, Stateflow, and Simscape. You can fully specify, test, and analyze the behavior of a component using the model-based design process.

For more information about the model-based systems engineering workflow within System Composer, see “Compose and Analyze Systems Using Architecture Models” on page 2-2.

In this tutorial, you will perform these steps on the robot arm architecture model.

Robot Arm Architecture Model

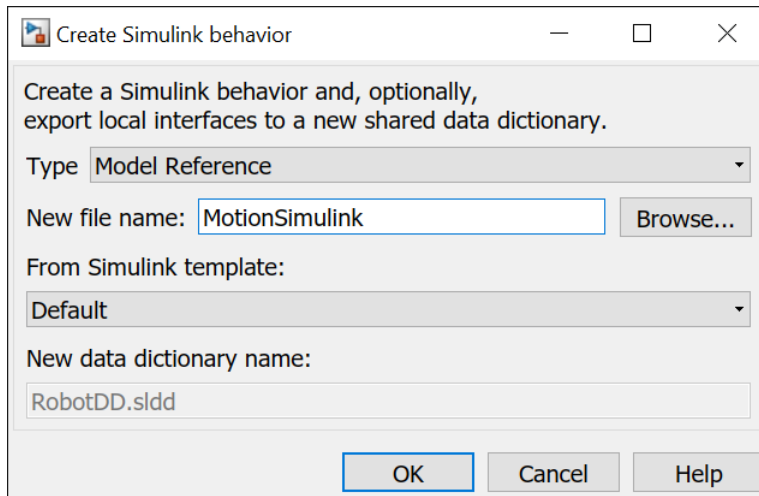
Open the architecture model of a robot arm that consists of sensors, motion actuators, and a planning algorithm. You can use System Composer to view the interfaces and manage the requirements for this model.




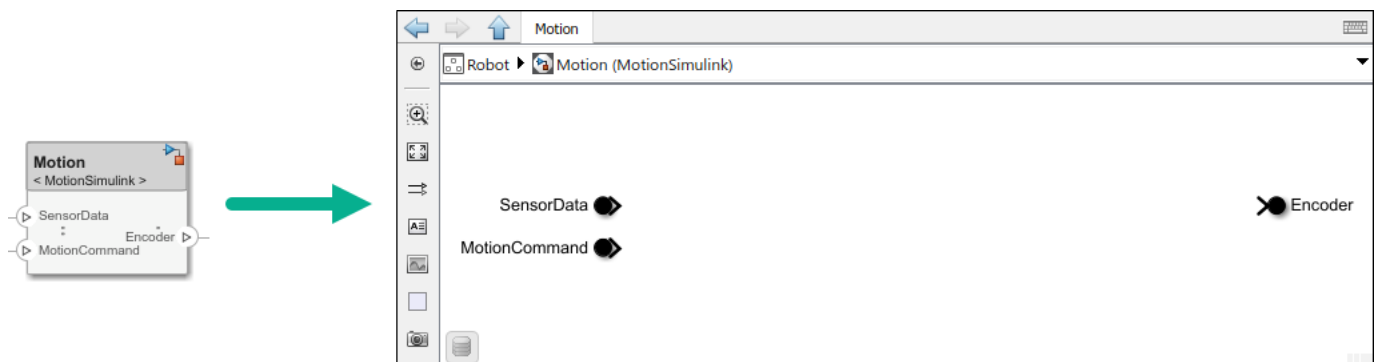
Reference Simulink Behavior Model in Component

When a component does not require further architectural decomposition, you can enable model simulation and an end-to-end workflow. To enable model simulation, implement Simulink behaviors for components. You can associate a Simulink model with a component or link to an existing Simulink model or subsystem.

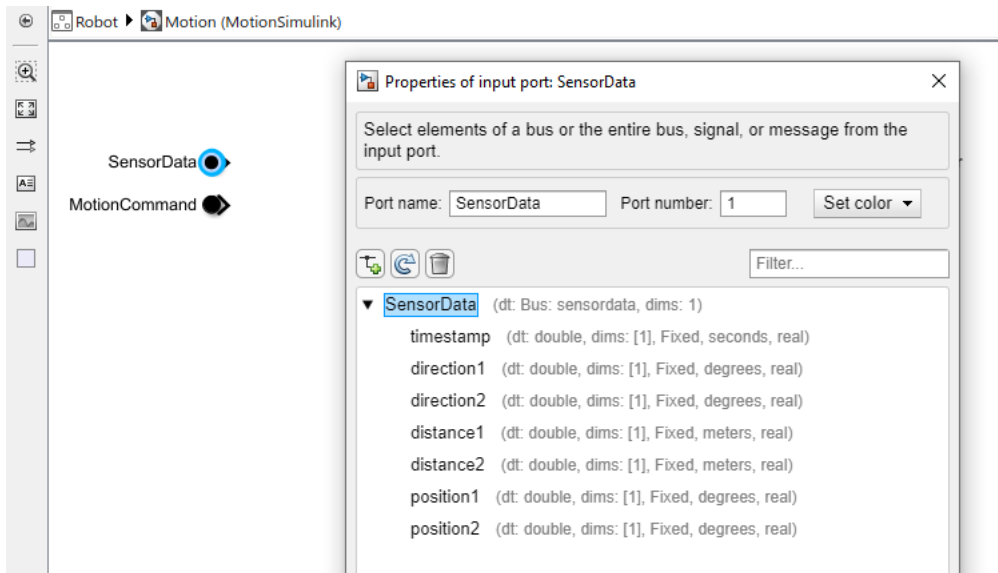
- 1 Navigate to **Modeling > Create Simulink Behavior**. Alternatively, right-click the `Motion` component and select `Create Simulink Behavior`.
- 2 From the **Type** list, select `Model Reference`. Provide the model name `MotionSimulink`. The default name is the name of the component.



- 3 A new Simulink model file with the provided name is created in the current folder. 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  icon on the component indicates that the component has a Simulink behavior.



- 4 To view the interfaces on the **SensorData** port converted into Simulink bus elements, double-click the port in Simulink.

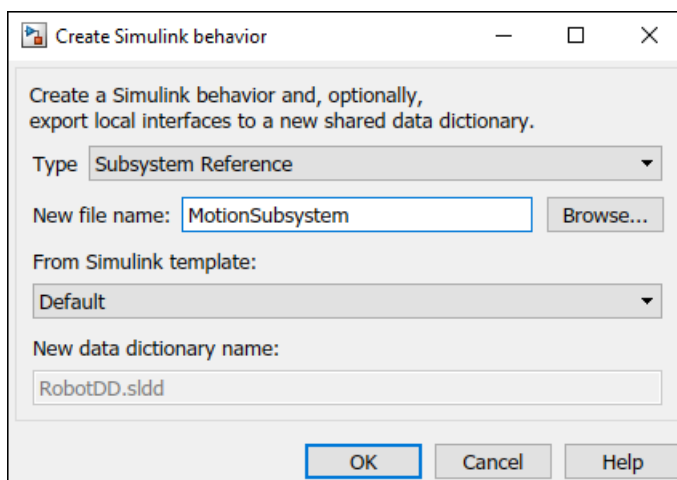



- 5 To remove model behavior, right-click the linked **Motion** component and select **Inline Model**.

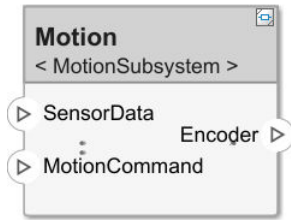
You can also link to an existing Simulink behavior model from a System Composer component, provided that the component is not already linked to a reference architecture. Right-click the component and select **Link to Model**. You can type or browse for the name of a Simulink model.

You can also link a referenced Simulink subsystem behavior to a component. Use subsystem references to author Simulink or Simscape behaviors with physical ports, connections, and blocks.

- 1 Navigate to **Modeling > Create Simulink Behavior**. Alternatively, right-click the **Motion** component and select **Create Simulink Behavior**
- 2 From the **Type** list, select **Subsystem Reference**. Provide the model name **MotionSubsystem**. The default name is the name of the component.



- 3 A new Simulink subsystem file with the provided name is created in the current folder. The root-level ports of the Simulink subsystem reflect the ports of the component. The component in the architecture model is linked to the Simulink subsystem. The  icon on the component indicates that the component has a Simulink subsystem behavior.



Add Stateflow Chart Behavior to Component

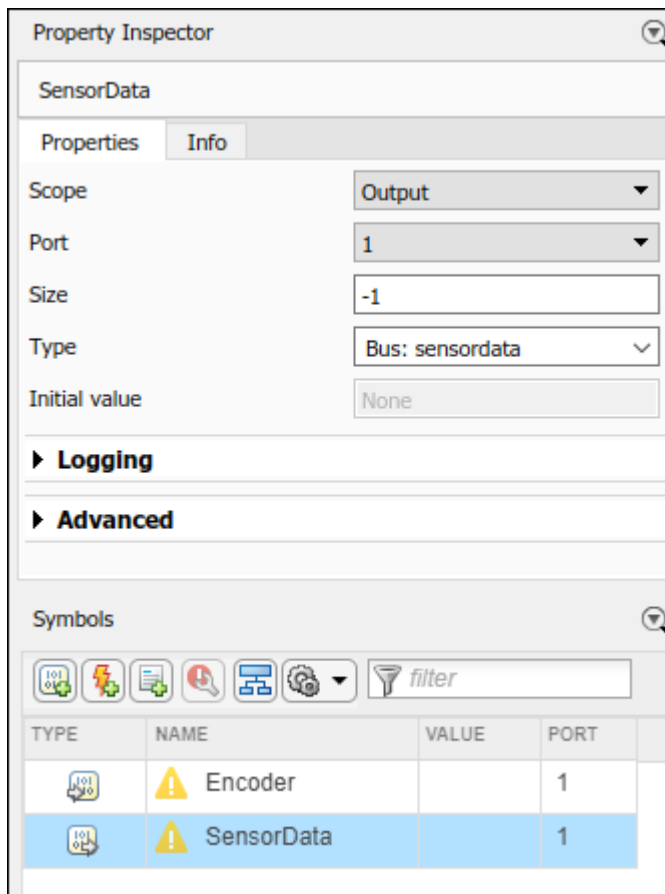
To implement event-based modeling with state machines, add Stateflow chart behavior to a component. State charts consist of a finite set of states with transitions between them to capture the modes of operation for the component. This functionality requires a Stateflow license.

A System Composer component with stereotypes, interfaces, requirement links, and ports, is preserved when you add Stateflow Chart behavior.

- 1 Right-click the Sensor component and select **Create Stateflow Chart Behavior**. Alternatively, select the Sensor component, then navigate to **Modeling > Create Stateflow Chart Behavior**.
- 2 Double-click **Sensor**, which has the Stateflow icon. From the **Modeling** menu, click **Symbols Pane** to view the Stateflow symbols. The input port **Encoder** appears as input data in the symbols pane and the output port **SensorData** appears as output data.



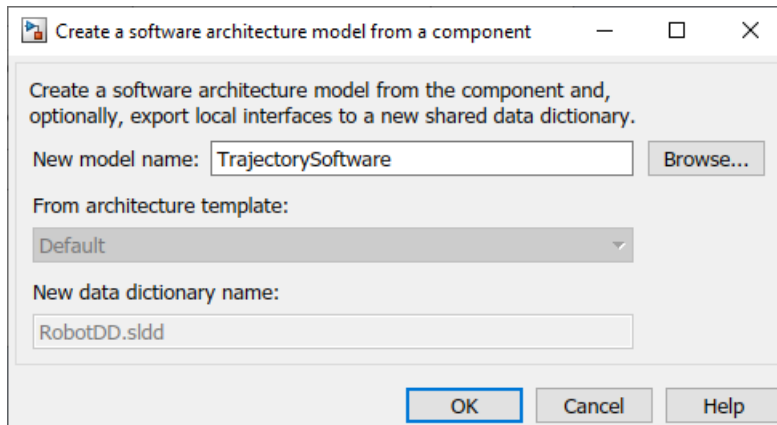
- 3 Select the **SensorData** output and view the interface in the **Property Inspector**. You can access this interface like a Simulink bus signal.



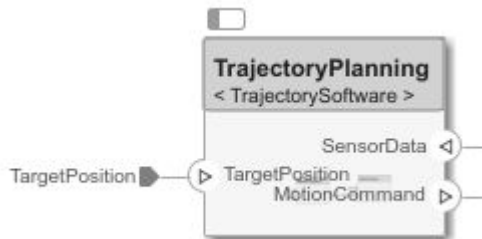
Design Software Architecture in Component

To design a software architecture, define function execution order, simulate, and generate code, create a software architecture from a System Composer component.

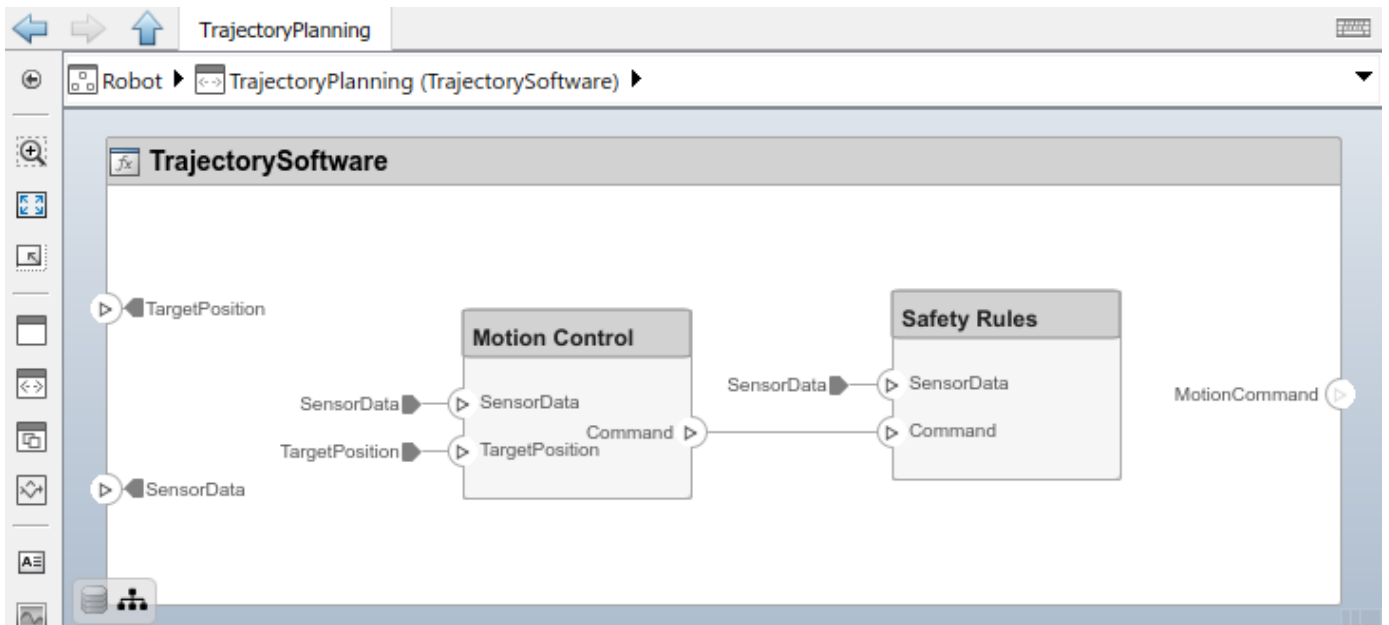
- 1 Rename the Trajectory Planning component to TrajectoryPlanning so that it is a valid C variable name.
- 2 Right-click the TrajectoryPlanning component and select Create Software Architecture Model, or, navigate to **Modeling > Create Software Architecture Model**.
- 3 Specify the name of the software architecture as TrajectorySoftware. Click **OK**.



- 4 The software architecture model `TrajectorySoftware.slx` is referenced from the `TrajectoryPlanning` component.



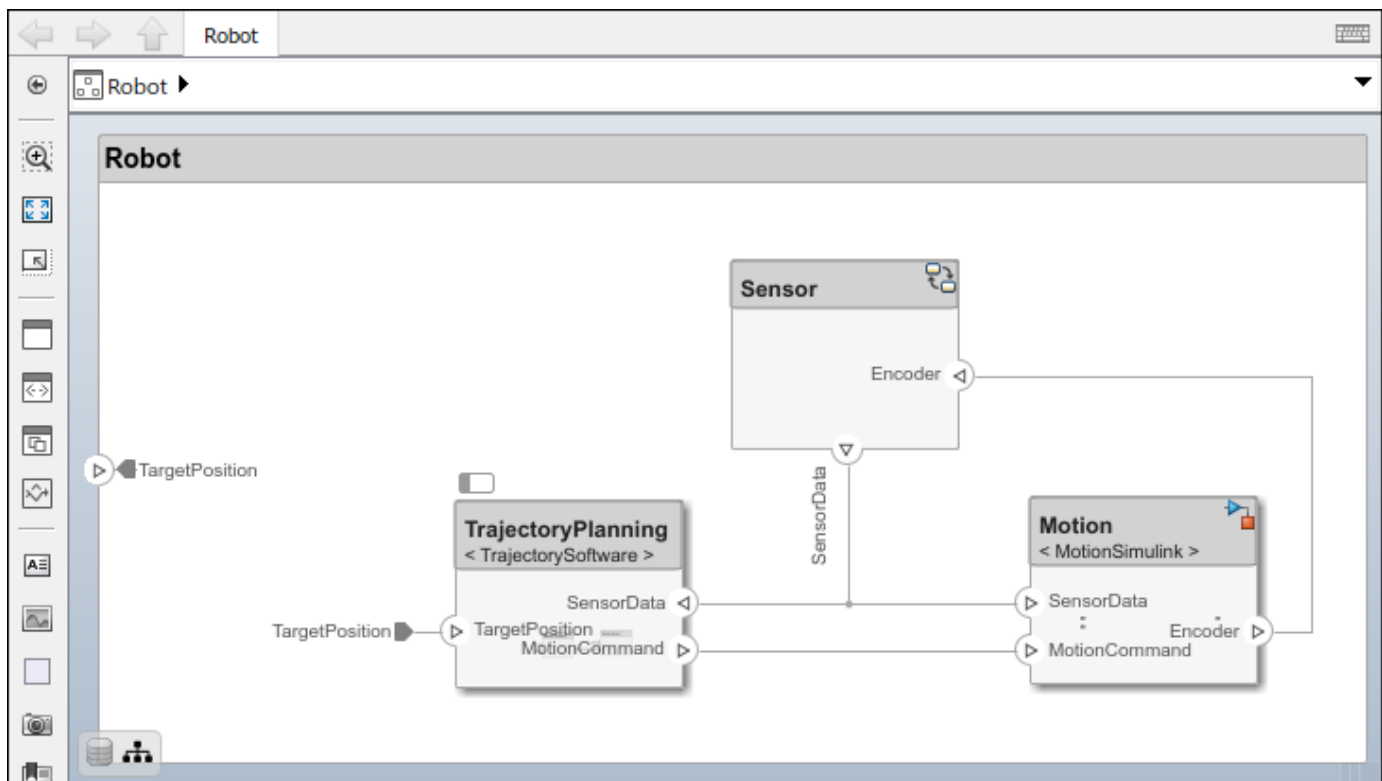
- 5 Double-click the `TrajectoryPlanning` component to interact with the `TrajectorySoftware` software component.



Represent System Interaction Using Sequence Diagrams

To represent the interaction between structural elements of an architecture as a sequence of message exchanges, use a sequence diagram in the **Architecture Views Gallery**.

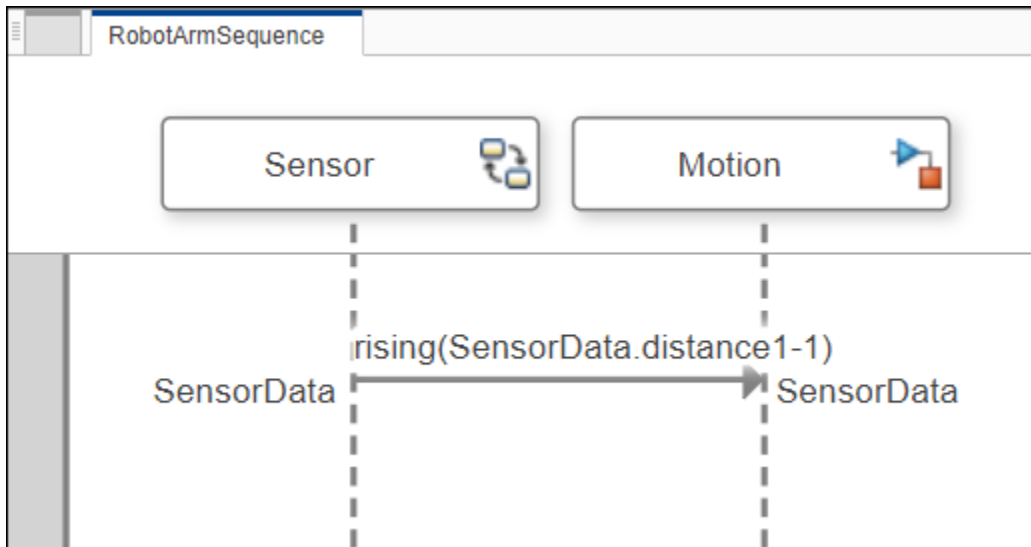
Observe the robot arm architecture model consisting of components, ports, connections, and behaviors. The model simulation results must match the interactions within the sequence diagrams.



- 1 Create a new sequence diagram by navigating to **Modeling > Sequence Diagram**. The **Architecture Views Gallery** opens. To create a new sequence diagram, click **+ New > Sequence Diagram**.
- 2 A new sequence diagram called SequenceDiagram1 is created in the View Browser, and the **Sequence Diagram** tab becomes active. Under **Sequence Diagram Properties**, rename the sequence diagram RobotArmSequence.
- 3 Select **Component > Add Lifeline** **+** to add a lifeline. A new lifeline with no name is created and is indicated by a dotted line.
- 4 Click the down arrow and select Sensor. Add a second lifeline named Motion.
- 5 Select the vertical dotted line for the Sensor lifeline. Click and drag to the Motion lifeline. In the **To** box, start to type Sensordata and choose SensorData from the drop down menu. A message is created from the **SensorData** port on the Sensor component to the **SensorData** port on the Motion component.
- 6 Click on the message to see where to place the message condition. Enter a message trigger condition in the form:


```
rising(SensorData.distance1-1)
```

The signal name is a data element on a data interface. The message will be recognized at the zero-crossing event when the value of `SensorData.distance1` rises to 1.



See Also

More About

- "Create Architecture Model with Interfaces and Requirement Links" on page 2-5
- "Extend Architectural Design Using Stereotypes" on page 2-19
- "Analyze Architecture Model with Analysis Function" on page 2-25
- "Inspect Components in Custom Architecture Views" on page 2-30
- "System Composer Concepts" on page 3-2

System Composer Terminology

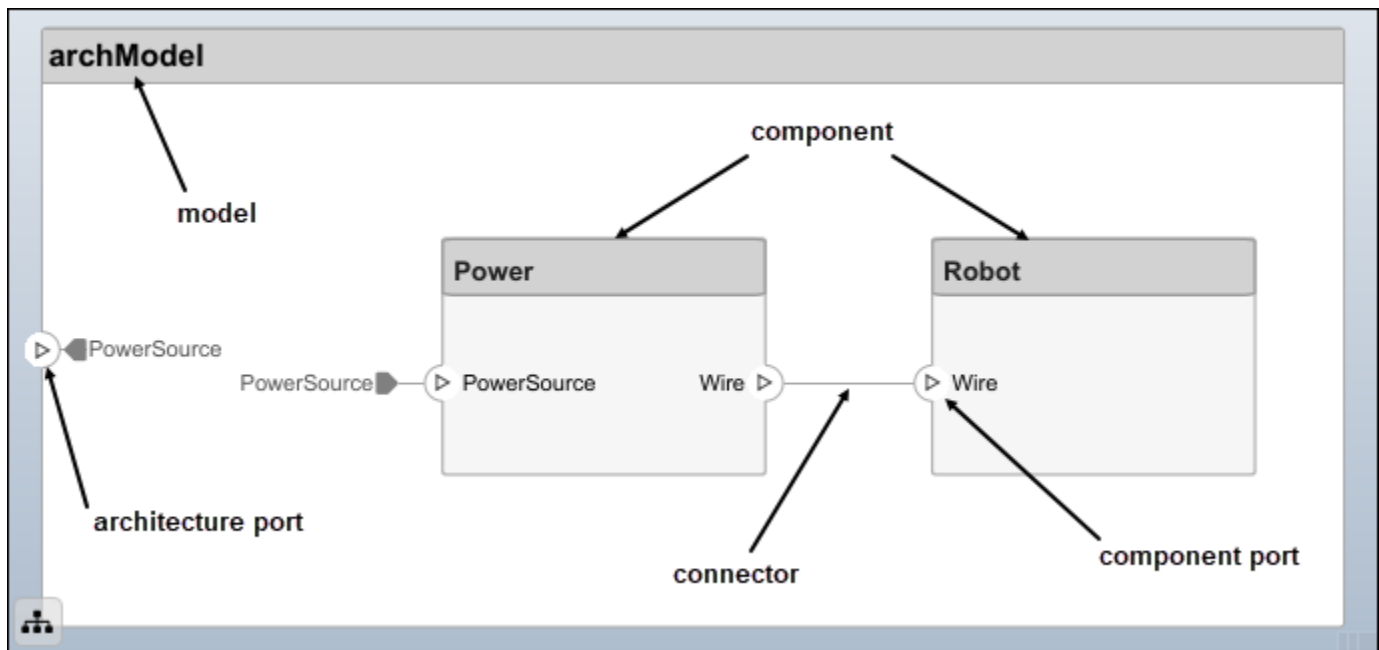
System Composer Concepts

System Composer combines concepts from systems engineering with concepts from Simulink. This page defines these concepts and their respective applications help you understand how these domains overlap in System Composer. Based on your architectural modeling goal, review the corresponding section to learn more about key concepts associated with that goal.

Use this page to learn about System Composer concepts and how they apply to systems engineering design. Each section defines a concept, explains how the concept is used in System Composer, then links to more information in the documentation.

Author Architecture Models

An architecture model in System Composer consists of the common Simulink constructions: components, ports, and connectors. An architecture represents the system of components.

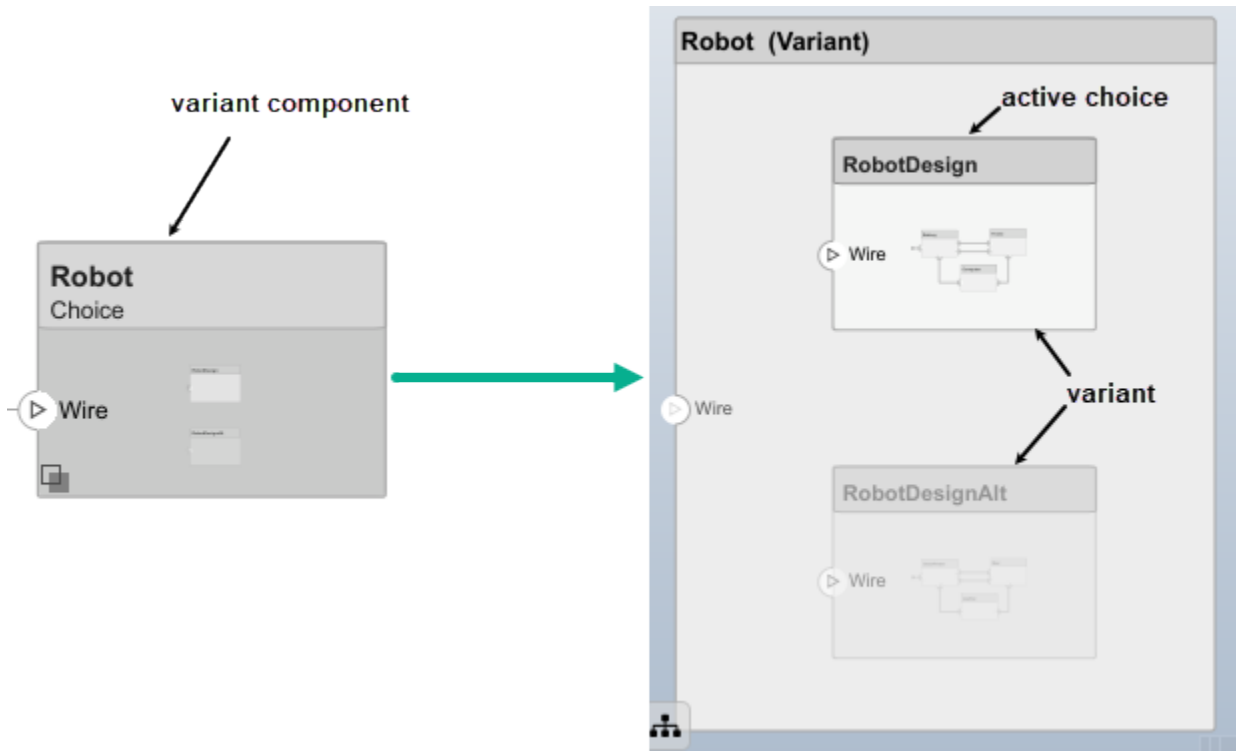


Term	Definition	Application	More Information
architecture	A System Composer architecture represents a system of components and how they interface with each other structurally and behaviorally.	Different types of architectures describe different aspects of systems. You can use views to visualize a subset of components in an architecture. You can define parameters on the architecture level using the Parameter Editor .	<ul style="list-style-type: none"> “Compose Architectures Visually” “Author Parameters in System Composer Using Parameter Editor”

Term	Definition	Application	More Information
model	A System Composer model is the file that contains architectural information, including components, ports, connectors, interfaces, and behaviors.	Perform operations on a model: <ul style="list-style-type: none"> • Extract the root-level architecture contained in the model. • Apply profiles. • Link interface data dictionaries. • Generate instances from model architecture. A System Composer model is stored as an SLX file.	“Create Architecture Model with Interfaces and Requirement Links” on page 2-5
component	A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of an architecture. A component defines an architectural element, such as a function, a system, hardware, software, or other conceptual entity. A component can also be a subsystem or subfunction.	Represented as a block, a component is a part of an architecture model that can be separated into reusable artifacts. Transfer information between components with: <ul style="list-style-type: none"> • Port interfaces using the Interface Editor • Parameters using the Parameter Editor 	“Components”
port	A port is a node on a component or architecture that represents a point of interaction with its environment. A port permits the flow of information to and from other components or systems.	There are different types of ports: <ul style="list-style-type: none"> • <i>Component ports</i> are interaction points on the component to other components. • <i>Architecture ports</i> are ports on the boundary of the system, whether the boundary is within a component or the overall architecture model. 	“Ports”
connector	Connectors are lines that provide connections between ports. Connectors describe how information flows between components or architectures.	A connector allows two components to interact without defining the nature of the interaction. Set an interface on a port to define how the components interact.	“Connections”

Manage Variants

Create variant components and implement multiple design alternatives or variants, chosen based on programmatic rules. Add variant choices to any component to make a variant component. The active choice represents the original component.



Term	Definition	Application	More Information
variant	A variant is one of many structural or behavioral choices in a variant component.	Use variants to quickly swap different architectural designs for a component while performing analysis.	“Create Variants”
variant control	A variant control is a string that controls the active variant choice.	Set the variant control to programmatically control which variant is active.	“Set Variant Control Condition”

Manage Interfaces

Assign interfaces to ports using the **Interface Editor** in **Dictionary View**. Use an Adapter block to reconcile differences between interfaces on a connector between ports.

The screenshot shows the System Composer interface with an architectural model and an 'Interfaces' panel. The model includes components like PowerSource, Power, Robot, and an adapter. The 'Interfaces' panel is in 'Dictionary View' and contains the following table:

interface data dictionary	Type	Dimensions	Units
archDictionary.sidd			
Charger ← data interface			
Voltage (VoltageType)	VoltageType	1	W
Wiring ← data element	double	1	m
RobotPower			
RobotVoltage (VoltageType)	VoltageType	1	W
Wires	double	1	m
VoltageType ← value type	double	1	W

Manage owned interfaces local to a port using the **Interface Editor** in **Port Interface View**.

The screenshot shows the same architectural model as the previous image, but the 'Interfaces' panel is now in 'Port Interface View'. An arrow points to a port on the 'PowerSource' component in the model. The 'Interfaces' panel shows the following table:

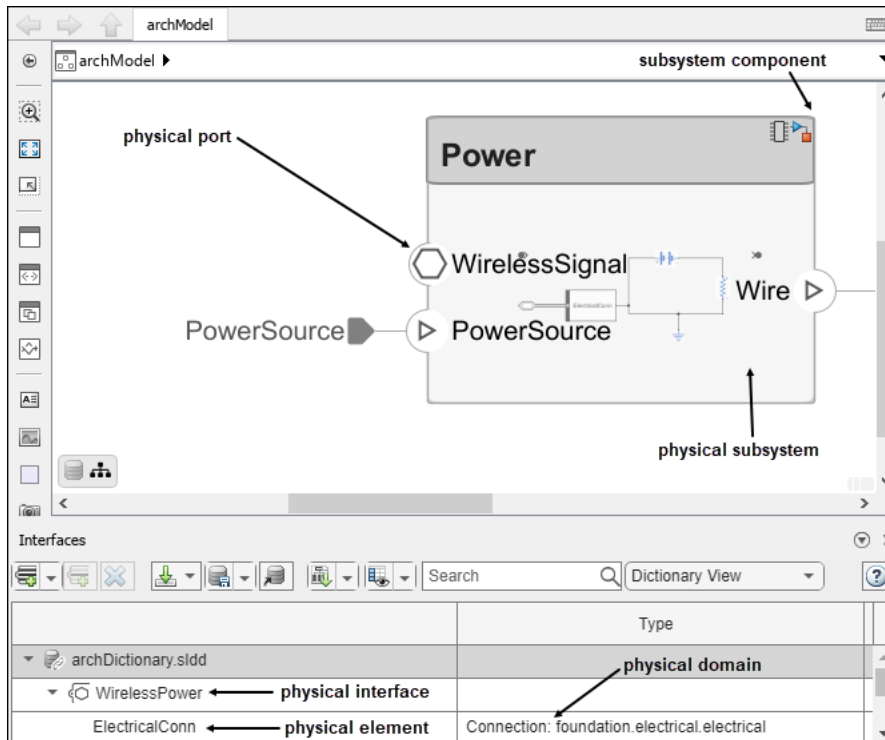
	Type	Dimensions	Units
PowerSource			
Life ← owned element	double	1	days

Term	Definition	Application	More Information
interface data dictionary	An interface data dictionary is a consolidated list of all the interfaces and value types in an architecture and where they are used.	Local interfaces on a System Composer model can be saved in an interface data dictionary using the Interface Editor . You can reuse interface dictionaries between models that need to use a given set of interfaces, elements, and value types. Linked data dictionaries are stored in separate SLDD files.	<ul style="list-style-type: none"> • “Manage Interfaces with Data Dictionaries” • “Reference Data Dictionaries”
data interface	A data interface defines the kind of information that flows through a port. The same interface can be assigned to multiple ports. A data interface can be composite, meaning that it can include data elements that describe the properties of an interface signal.	Data interfaces represent the information that is shared through a connector and enters or exits a component through a port. Use the Interface Editor to create and manage data interfaces and data elements and store them in an interface data dictionary for reuse between models.	<ul style="list-style-type: none"> • “Create Architecture Model with Interfaces and Requirement Links” on page 2-5 • “Define Port Interfaces Between Components”
data element	A data element describes a portion of an interface, such as a communication message, a calculated or measured parameter, or other decomposition of that interface.	Data interfaces are decomposed into data elements: <ul style="list-style-type: none"> • Pins or wires in a connector or harness. • Messages transmitted across a bus. • Data structures shared between components. 	<ul style="list-style-type: none"> • “Create Interfaces” • “Assign Interfaces to Ports”
value type	A value type can be used as a port interface to define the atomic piece of data that flows through that port and has a top-level type, dimension, unit, complexity, minimum, maximum, and description.	You can also assign the type of data elements in data interfaces to value types. Add value types to data dictionaries using the Interface Editor so that you can reuse the value types as interfaces or data elements.	“Create Value Types as Interfaces”
owned interface	An owned interface is an interface that is local to a specific port and not shared in a data dictionary or the model dictionary.	Create an owned interface to represent a value type or data interface that is local to a port.	“Define Owned Interfaces Local to Ports”

Term	Definition	Application	More Information
adapter	An adapter helps connect two components with incompatible port interfaces by mapping between the two interfaces. An adapter can act as a unit delay or rate transition. You can also use an adapter for bus creation. Use the Adapter block to implement an adapter.	<p>With an adapter, you can perform functions on the “Interface Adapter” dialog box:</p> <ul style="list-style-type: none"> • Create and edit mappings between input and output interfaces. • Apply an interface conversion <code>UnitDelay</code> to break an algebraic loop. • Apply an interface conversion <code>RateTransition</code> to reconcile different sample time rates for reference models. • Apply an interface conversion <code>Merge</code> to merges two or more message or signal lines. • When output interfaces are undefined, you can use input interfaces in bus creation mode to author owned output interfaces. 	<ul style="list-style-type: none"> • “Interface Adapter” • Adapter

Author Physical Models

Author physical models in System Composer using subsystem components. A subsystem component is a Simulink subsystem that is part of the parent System Composer architecture model. You can add Simscape behavior to a subsystem component using physical ports, connectors, and interfaces. For more information, see “Author Model Behavior” on page 3-20.



Term	Definition	Application	More Information
physical subsystem	A physical subsystem is a Simulink subsystem with Simscape connections.	A physical subsystem with Simscape connections uses a physical network approach suited for simulating systems with real physical components and represents a mathematical model.	“Implement Component Behavior Using Simscape”
physical port	A physical port represents a Simscape physical modeling connector port called a Connection Port.	Use physical ports to connect components in an architecture model or to enable physical systems in a Simulink subsystem.	“Define Physical Ports on Component”
physical connector	A physical connector can represent a nondirectional conserving connection of a specific physical domain. Connectors can also represent physical signals.	Use physical connectors to connect physical components that represent features of a system to simulate mathematically.	“Architecture Model with Simscape Behavior for a DC Motor”

Term	Definition	Application	More Information
physical interface	A physical interface defines the kind of information that flows through a physical port. The same interface can be assigned to multiple ports. A physical interface is a composite interface equivalent to a <code>Simulink.ConnectionBus</code> object that specifies any number of <code>Simulink.ConnectionElement</code> objects.	Use a physical interface to bundle physical elements to describe a physical model using at least one physical domain.	"Specify Physical Interfaces on Ports"
physical element	A physical element describes the decomposition of a physical interface. A physical element is equivalent to a <code>Simulink.ConnectionElement</code> object.	Define the Type of a physical element as a physical domain to enable use of that domain in a physical model.	"Describe Component Behavior Using Simscape"

Extend Architectural Elements

Create a profile in the **Profile Editor** and add stereotypes to it with properties. Apply the stereotype to a component, and set the property value in the **Property Inspector**.

The screenshot displays the System Composer interface with three main panels:

- Profile Browser:** Shows a tree view with 'RobotProfile' (labeled as 'profile') and 'ElectricalComponent' (labeled as 'stereotype') under it.
- Stereotype Properties:** Shows configuration for the 'ElectricalComponent' stereotype. The 'Name' is 'ElectricalComponent', 'Applies to' is 'Component', and 'Base stereotype' is '<nothing>'. Below is a table of default stereotypes for composition:

Property name	Type	Name	Unit	Default
1 Power	double	n/a	W	0

- Property Inspector:** Shows the 'Component' selected in the model. The 'Info' tab is active, displaying a table of properties:

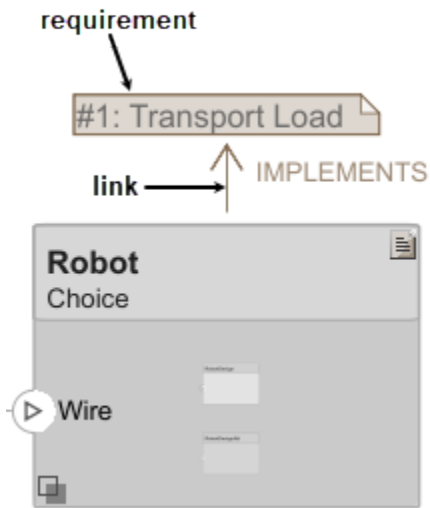
NAME	VALUE
Main	
Name	Power
Stereotype	Add..
ElectricalComponent	
Power	150 W
Parameters	

Arrows in the Property Inspector point to the 'Power' property name (labeled 'property') and its value '150 W' (labeled 'property value').

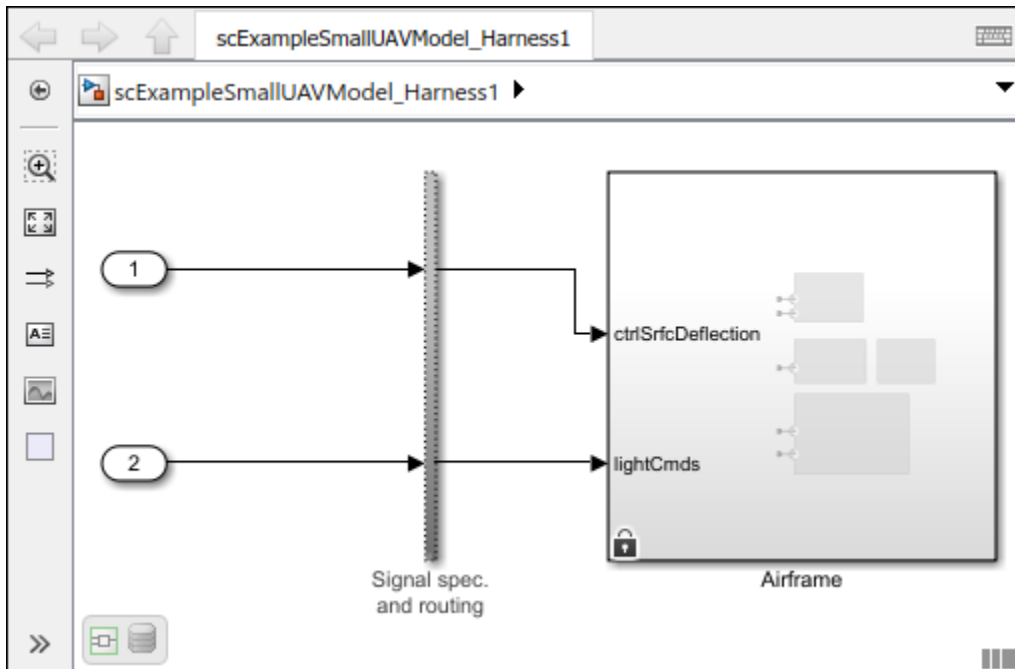
Term	Definition	Application	More Information
stereotype	A stereotype is a custom extension of the modeling language. Stereotypes provide a mechanism to extend the architecture language elements by adding domain-specific metadata.	Apply stereotypes to model elements such as root-level architecture, component architecture, connectors, ports, data interfaces, value types, functions, requirements, and links. Functions only apply to software architectures. You must have a Requirements Toolbox license to apply stereotypes to requirements and links. A model element can have multiple stereotypes. Stereotypes provide model elements with a common set of property fields, such as mass, cost, and power.	“Extend Architectural Design Using Stereotypes” on page 2-19
property	A property is a field in a stereotype. You can specify property values for each element to which the stereotype is applied.	Use properties to store quantitative characteristics, such as weight or speed, that are associated with a model element. Properties can also be descriptive or represent a status. You can view and edit the properties of each element in the architecture model using the Property Inspector .	<ul style="list-style-type: none"> • “Set Properties” on page 2-23 • “Add Properties with Stereotypes” • “Set Properties for Analysis”
profile	A profile is a package of stereotypes that you can use to create a self-consistent domain of element types.	Author profiles and apply profiles to a model using the Profile Editor . You can store stereotypes for a project in one or several profiles. When you save profiles, they are stored in XML files.	<ul style="list-style-type: none"> • “Define Profiles and Stereotypes” • “Use Stereotypes and Profiles”

Manage and Verify Requirements

In the Requirements Perspective, you can create, manage, and allocate requirements. View the requirements on the architecture model. This functionality requires a Requirements Toolbox license.



Use Simulink Test to create a test harness for a System Composer component to validate simulation results and verify design in the **Test Manager**. This functionality requires a Simulink Test license.

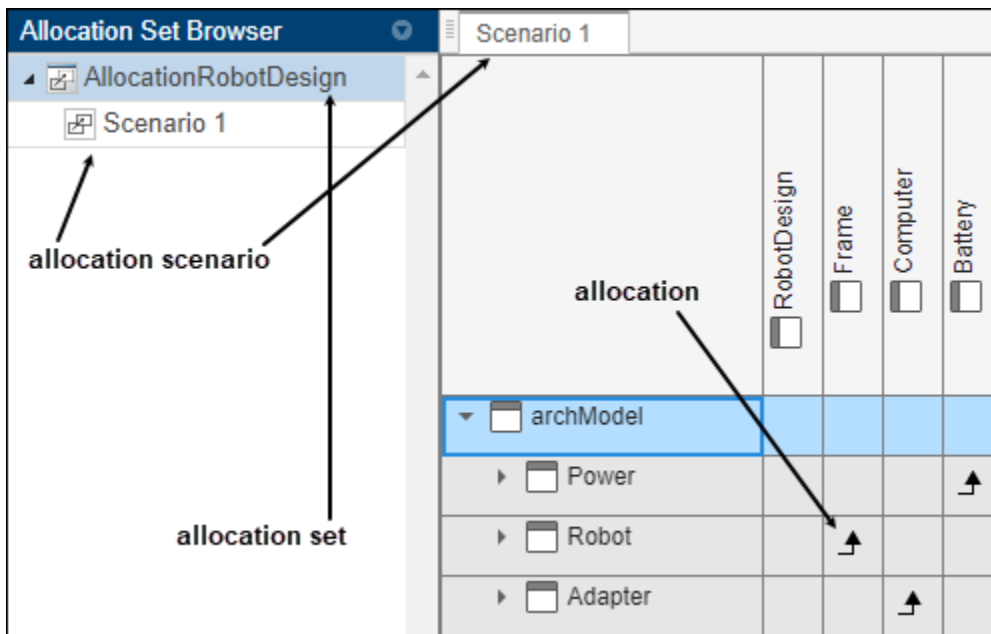


Term	Definition	Application	More Information
requirements	Requirements are a collection of statements describing the desired behavior and characteristics of a system. Requirements ensure system design integrity and are achievable, verifiable, unambiguous, and consistent with each other. Each level of design should have appropriate requirements.	To enhance traceability of requirements, link system, functional, customer, performance, or design requirements to components and ports. Link requirements to each other to represent derived or allocated requirements. Manage requirements from the Requirements Perspective on an architecture model or through custom views. Assign test cases to requirements using the Test Manager for verification and validation.	<ul style="list-style-type: none"> • “Link and Trace Requirements” • “Establish Traceability Between Architectures and Requirements”
requirement set	A requirement set is a collection of requirements. You can structure the requirements hierarchically and link them to components or ports.	Use the Requirements Editor to edit and refine requirements in a requirement set. Requirement sets are stored in SLREQX files. You can create a new requirement set and author requirements using Requirements Toolbox, or import requirements from supported third-party tools.	<ul style="list-style-type: none"> • “Allocate and Trace Requirements from Design to Verification” • “Manage Requirements”
requirement link	A link is an object that relates two model-based design elements. A requirement link is a link where the destination is a requirement. You can link requirements to components or ports.	View links using the Requirements Perspective in System Composer. Select a requirement in the Requirements Browser to highlight the component or the port to which the requirement is assigned. Links are stored externally as SLMX files.	<ul style="list-style-type: none"> • “Create Architecture Model with Interfaces and Requirement Links” on page 2-5 • “Update Reference Requirement Links from Imported File”

Term	Definition	Application	More Information
test harness	A test harness is a model that isolates the component under test with inputs, outputs, and verification blocks configured for testing scenarios. You can create a test harness for a model component or for a full model. A test harness gives you a separate testing environment for a model or a model component.	Create a test harness for a System Composer component to validate simulation results and verify design. To edit the interfaces while you are testing the behavior of a component in a test harness, use the Interface Editor .	<ul style="list-style-type: none"> “Verify and Validate Requirements” “Create a Test Harness” (Simulink Test)

Allocate Architecture Models

In the **Allocation Editor**, allocate components between two architecture models, based on a dependency or a directed relationship.



Term	Definition	Application	More Information
allocation	An allocation establishes a directed relationship from architectural elements — components, ports, and connectors — in one model to architectural elements in another model.	Resource-based allocation allows you to allocate functional architectural elements to logical architectural elements and logical architectural elements to physical architectural elements.	<ul style="list-style-type: none"> “Create and Manage Allocations Interactively” “Create and Manage Allocations Programmatically”

Term	Definition	Application	More Information
allocation scenario	An allocation scenario contains a set of allocations between a source and a target model.	Allocate between model elements in an allocation scenario. The default allocation scenario is called Scenario 1.	"Systems Engineering Approach for SoC Applications"
allocation set	An allocation set consists of one or more allocation scenarios that describe various allocations between a source and a target model.	Create an allocation set with allocation scenarios in the Allocation Editor . Allocation sets are saved as MLDATX files.	<ul style="list-style-type: none"> "Establish Traceability Between Architectures and Requirements" "Allocate Architectures in Tire Pressure Monitoring System"

Create Custom Views

Apply a view filter to generate an element group of components for the view in the **Architecture Views Gallery**.

The screenshot displays the System Composer interface. At the top, a view titled "Electrical Elements" is shown, containing an "element group" named "Power". The "Power" element group includes components "PowerSource", "WirelessSignal", and "Wire". A label "view" points to the "Electrical Elements" view, and a label "element group" points to the "Power" element group.

Below the view, the "View Configurations" section is visible, showing "Requirement Links" and "FILTER" options. The "FILTER" section includes "Add Component Filter", "Add Port Filter", "Apply", "Revert", "Auto Apply", and "Mode" (Basic, Code). A label "filter" points to the "Filter" section.

The "COMPONENT FILTER" section shows the following configuration:

- Select: Components
- Where: Stereotype
- isa: RobotProfile.ElectricalComp...

Term	Definition	Application	More Information
view	A view shows a customizable subset of elements in a model. Views can be filtered based on stereotypes or names of components, ports, and interfaces, along with the name, type, or units of an interface element. Create views by adding elements manually. Views create a simplified way to work with complex architectures by focusing on certain parts of the architectural design.	<p>You can use different types of views to represent the system. Switch between a component diagram, component hierarchy, or architecture hierarchy. For software architectures, you can switch to a class diagram view.</p> <p>A viewpoint represents a stakeholder perspective that specifies the contents of the view.</p>	“Modeling System Architecture of Keyless Entry System”
element group	An element group is a grouping of components in a view.	Use element groups to programmatically populate a view.	<ul style="list-style-type: none"> • “Create Architecture Views Interactively” • “Create Architectural Views Programmatically”
query	A query is a specification that describes certain constraints or criteria to be satisfied by model elements.	Use queries to search elements with constraint criteria and to filter views.	“Find Elements in Model Using Queries”
component diagram	A component diagram represents a view with components, ports, and connectors based on how the model is structured.	Component diagrams allow you to programmatically or manually add and remove components from the view.	“Inspect Components in Custom Architecture Views” on page 2-30

Term	Definition	Application	More Information
hierarchy diagram	You can visualize a hierarchy diagram as a view with components, ports, reference types, component stereotypes, and stereotype properties.	<p>There are two types of hierarchy diagrams:</p> <ul style="list-style-type: none"> <i>Component hierarchy diagrams</i> display components in tree form with parents above children. In a component hierarchy view, each referenced model is represented as many times as it is used. <i>Architecture hierarchy diagrams</i> display unique component architecture types and their relationships using composition connections. In an architecture hierarchy view, each referenced model is represented only once. 	“Display Component Hierarchy and Architecture Hierarchy Using Views”

Analyze Architecture Models

Create an analysis function to analyze power consumption in the RobotDesign architecture model.

```
function RobotDesign_1(instance,varargin)
if instance.isComponent() && ~isempty(instance.Components)...
    && instance.hasValue('RobotProfile.ElectricalComponent.Power')
        sysComponent_power = 0;
        for child = instance.Components
            if child.hasValue('RobotProfile.ElectricalComponent.Power')
                comp_power = child.getValue('RobotProfile.ElectricalComponent.Power');
                sysComponent_power = sysComponent_power + comp_power;
            end
        end
    end
instance.setValue('RobotProfile.ElectricalComponent.Power',sysComponent_power);
end
```

Analyze the robot design using the analysis function to determine total power usage.

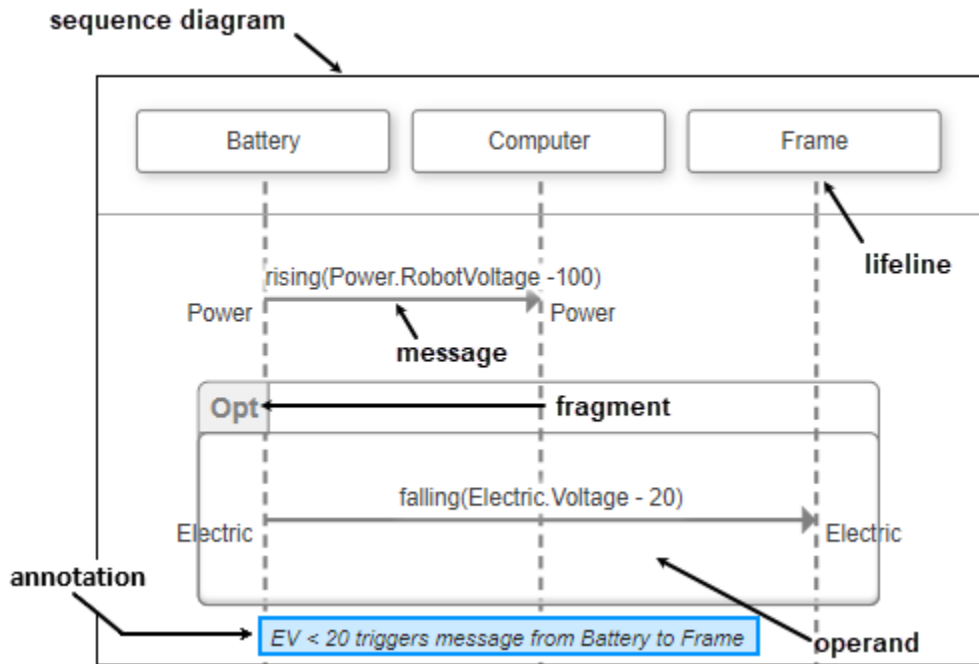
Instances	Power
RobotDesign	380
Battery	130
Computer	200
Frame	50

INSTANCE PROPERTIES		
ComponentInstance: RobotDesign		
Property	Value	Units
RobotProfile.ElectricalComponent		
Power	380	W

Term	Definition	Application	More Information
analysis	Analysis is a method for quantitatively evaluating an architecture for certain characteristics. Static analysis analyzes the structure of the system. Static analysis uses an analysis function and parametric values of properties captured in the system model.	Use analyses to calculate overall reliability, mass roll-up, performance, or thermal characteristics of a system, or to perform a SWaP analysis.	<ul style="list-style-type: none"> • “Analyze Architecture Model with Analysis Function” on page 2-25 • “Analyze Architecture” • “Simple Roll-Up Analysis Using Robot System with Properties”
analysis function	An analysis function is a MATLAB function that computes values necessary to evaluate the architecture using the properties of each element in the model instance.	Use an analysis function to calculate the result of an analysis.	<ul style="list-style-type: none"> • “Analysis Function Constructs” • “Write Analysis Function”
instance model	An instance model is a collection of instances.	You can update an instance model with changes to a model, but the instance model will not update with changes in active variants or model references. You can use an instance model, saved in a MAT file, of a System Composer architecture model for analysis.	“Run Analysis Function”
instance	An instance is an occurrence of an architecture model element at a given point in time.	An instance freezes the active variant or model reference of the component in the instance model.	“Create a Model Instance for Analysis”

Author Sequence Diagrams

Create a sequence diagram in the **Architecture Views Gallery** to describe system interactions.

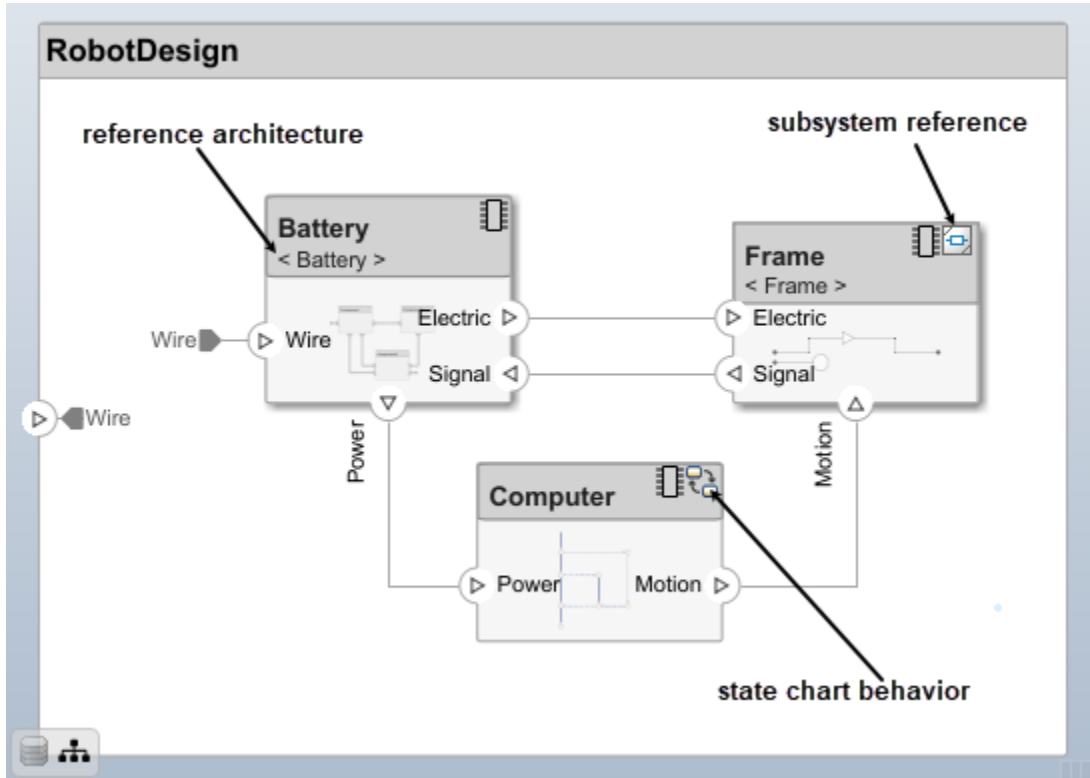


Term	Definition	Application	More Information
sequence diagram	A sequence diagram represents the expected interaction between structural elements of an architecture as a sequence of message exchanges.	Use sequence diagrams to describe how the parts of a system interact.	<ul style="list-style-type: none"> “Describe System Behavior Using Sequence Diagrams” “Use Sequence Diagrams with Architecture Models”
lifeline	A lifeline is represented by a head and a timeline that proceeds down a vertical dotted line.	The head of a lifeline represents a component in an architecture model.	“Add Lifelines and Messages”
message	A message sends information from one lifeline to another. Messages are specified with a message label.	A message label has a trigger and a constraint. A trigger determines whether the message occurs. A constraint determines whether the message is valid.	“Create Messages in Sequence Diagram”
annotation	An annotation describes the elements of a sequence diagram.	Use annotations to provide detailed explanations of elements or workflows captured by sequence diagrams.	“Use Annotations to Describe Elements of Sequence Diagram”
fragment	A fragment indicates how a group of messages within it execute or interact.	A fragment is used to model complex sequences, such as alternatives, in a sequence diagram.	“Author Sequence Diagram Fragments”

Term	Definition	Application	More Information
operand	An operand is a region in a fragment. Fragments have one or more operands depending on the kind of fragment. Operands can contain messages and additional fragments.	Each operand can include a constraint to specify whether the messages inside the operand execute. You can express the precondition of an operand as a MATLAB Boolean expression using the input signal of any lifeline.	"Add Fragments and Operands"

Author Model Behavior

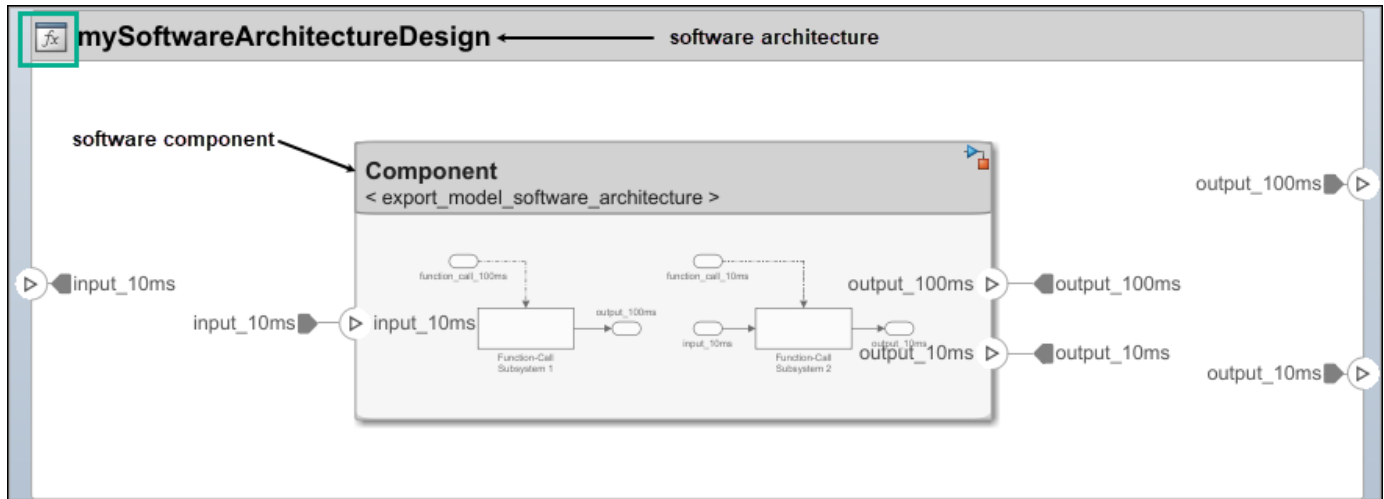
Use a reference component to decompose and reuse architectural components and Simulink model behaviors. Use a subsystem component or state chart to implement Simulink and Stateflow behaviors.



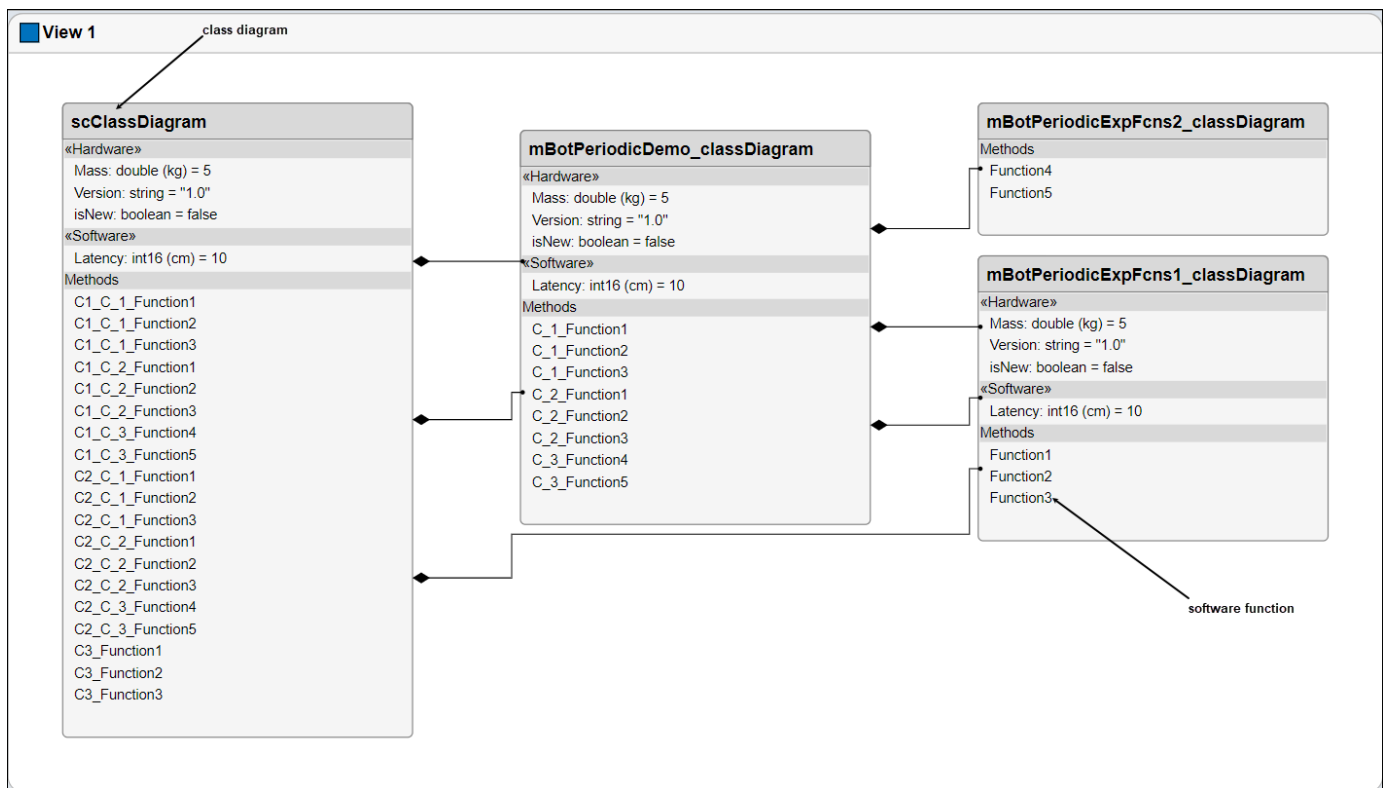
Term	Definition	Application	More Information
reference component	A reference component is a component whose definition is a separate architecture model, Simulink behavior model, or Simulink subsystem behavior. A reference component represents a logical hierarchy of other compositions.	<p>You can reuse compositions in the model using reference components. There are three types of reference components:</p> <ul style="list-style-type: none"> • <i>Model references</i> are Simulink models. • <i>Subsystem references</i> are Simulink subsystems. • <i>Architecture references</i> are System Composer architecture models or subsystems. 	<ul style="list-style-type: none"> • “Implement Component Behavior Using Simulink” • “Create Architecture Reference”
parameter	A parameter is an instance-specific value of a value type.	Parameters are available for inlined architectures and components. Parameters are also available for components linked to model references or architecture references that specify model arguments. You can specify independent values for a parameter on each component.	<ul style="list-style-type: none"> • “Author Parameters in System Composer Using Parameter Editor” • “Access Model Arguments as Parameters on Reference Components” • “Use Parameters to Store Instance Values with Components”
subsystem component	A subsystem component is a Simulink subsystem that is part of the parent System Composer architecture model.	Add Simulink subsystem behavior to a component to author a subsystem component in System Composer. You cannot synchronize and reuse subsystem components as Reference Component blocks because the component is part of the parent model.	<ul style="list-style-type: none"> • “Create Simulink Subsystem Behavior Using Subsystem Component” • “Create Simulink Subsystem Component”
state chart	A state chart diagram demonstrates the state-dependent behavior of a component throughout its state lifecycle and the events that can trigger a transition between states.	Add Stateflow chart behavior to describe a component using state machines. You cannot synchronize and reuse Stateflow chart behaviors as Reference Component blocks because the component is part of the parent model.	<ul style="list-style-type: none"> • “Implement Behaviors for Architecture Model Simulation” on page 2-37 • “Implement Component Behavior Using Stateflow Charts”

Design Software Architectures

Design a software architecture model, define the execution order of the functions from the components, simulate the design in the architecture level, and generate code.



View the software architecture diagram in a class diagram in the **Architecture Views Gallery**.



Term	Definition	Application	More Information
software architecture	A software architecture is a specialization of an architecture for software-based systems, including the description of software compositions, component functions, and their scheduling.	Use software architectures in System Composer to author software architecture models composed of software components, ports, and interfaces. Design your software architecture model, define the execution order of your component functions, simulate your design in the architecture level, and generate code.	<ul style="list-style-type: none"> • “Author Software Architectures” • “Simulate and Deploy Software Architectures”
software component	A software component is a specialization of a component for software entities, including its functions (entry points) and interfaces.	Implement a Simulink export-function, rate-based, or JMAAB model as a software component, simulate the software architecture model, and generate code.	<ul style="list-style-type: none"> • “Implement Behaviors for Architecture Model Simulation” on page 2-37 • “Create Software Architecture from Component”
software composition	A software composition is a diagram of software components and connectors that represents a composite software entity, such as a module or application.	Encapsulate functionality by aggregating or nesting multiple software components or compositions.	“Modeling Software Architecture of Throttle Position Control System”
function	A function is an entry point that can be defined in a software component.	You can apply stereotypes to functions in software architectures, edit sample times, and specify the function period using the Functions Editor .	“Author and Extend Functions for Software Architectures”
service interface	A service interface defines the functional interface between client and server components. Each service interface consists of one or more function elements.	Once you have defined a service interface in the Interface Editor , you can assign it to client and server ports using the Property Inspector . You can also use the Property Inspector to assign stereotypes to service interfaces.	<ul style="list-style-type: none"> • “Author Service Interfaces for Client-Server Communication” • <code>systemcomposer.interface.ServiceInterface</code>

Term	Definition	Application	More Information
function element	A function element describes the attributes of a function in a client-server interface.	<p>Edit the function prototype on a function element to change the number and names of inputs and outputs of the function. Edit function element properties as you would edit other interface element properties. Function argument types can include built-in types as well as bus objects. You can specify function elements to support:</p> <ul style="list-style-type: none"> • Synchronous execution <ul style="list-style-type: none"> – When the client calls the server, the function runs immediately and returns the output arguments to the client. • Asynchronous execution <ul style="list-style-type: none"> – When the client makes a request to call the server, the function is executed asynchronously based on the priority order defined in the Functions Editor and Schedule Editor and returns the output arguments to the client. 	systemcomposer.interface.FunctionElement
function argument	A function argument describes the attributes of an input or output argument in a function element.	You can set the properties of a function argument in the Interface Editor just as you would any value type: Type, Dimensions, Units, Complexity, Minimum, Maximum, and Description.	systemcomposer.interface.FunctionArgument
class diagram	A class diagram is a graphical representation of a static structural model that displays unique architecture types of the software components optionally with software methods and properties.	Class diagrams capture one instance of each referenced model and show relationships between them. Any component diagram view can be optionally represented as a class diagram for a software architecture model.	“Class Diagram View of Software Architectures”

See Also

More About

- “Compose and Analyze Systems Using Architecture Models” on page 2-2
- “Organize System Composer Files in Projects”
- “Simulate Mobile Robot with System Composer Workflow”
- “Modeling System Architecture of Small UAV”

External Websites

- Systems Engineering: Managing System Complexity

