Aiste Aleksandraviciene
Aurelijus Morkevicius, Ph.D.

MagicGrid®
Book of Knowledge

A Practical Guide to Systems Modeling
using MagicGrid from No Magic

Kaunas, 2018

Contents

Foreword by Enrique Krajmalnik

The last three years have seen significant change in the way products are designed. With the advent of smart products and delivery systems, the need for agile product development has never been greater. Companies are now looking at how they design and build as an integral part of the product life-cycle. Tooling is no longer an afterthought, and systems engineering is no longer a minor practice dominated by documents and spreadsheets with little to no traceability or change control. The transition to integrated tooling with a common, standard notation for system models is well underway, bringing a new level of rigor to systems engineering. Model-Based Systems Engineering (MBSE) is at the core of the new paradigm.

The ***MagicGrid Book of Knowledge (BoK)*** focuses on how practitioners implement MBSE. MagicGrid has been developed to provide a framework that can be easily adopted, implemented, and modified to grow with your modeling and systems engineering practice. This book teaches you how to use No Magic software to design and optimize the systems and products of today and tomorrow. It is the culmination of many years of research, and I am proud to put the No Magic name behind this effort.

Thank you to the No Magicians who made this happen, especially our solutions team who have engaged with customers over thousands of hours to learn, digest, develop, and teach this framework. I would also like to thank our partners and customers who contributed to peer review of this work prior to its release.

— Enrique Krajmalnik,
CTO, No Magic, Inc.

Foreword by Aurelijus Morkevicius

The ***MagicGrid BoK*** is a unique, model-based systems engineering experience. This book describes one of a million possible straightforward ways to develop a system model from A to Z. It is an answer to all the questions you may have, starting from "why" and ending with "how", which few of today's available sources of information can answer.

When I started MBSE training and consulting 10 years ago, the first thing that came to my mind was that there was a lack of an unambiguous approach to developing system models using SysML. Every customer was developing models differently. Moreover, every consultant used different approaches to train customers in this discipline. It should come as no surprise that SysML was (and still is) criticized for being method-agnostic. There was no other way forward than to initiate internal discussions within No Magic, and to start collecting and summarizing our experience in the form of a framework for architecting systems using a sort of "vanilla" SysML. The framework comes first, then adjusting to the language second. I am certain that this is the only approach that is 100 percent aligned with SysML. Extensions and customizations are possible; however, they are not necessary.

While collecting our knowledge, I did some scientific studies with a local university to prove my ideas were correct. Together with my team at No Magic, I wrote a couple of papers and delivered multiple presentations and tutorials at various events worldwide, e.g., the INCOSE International Symposium, INCOSE local chapter events, and the No Magic World Symposium, to name just a few. Publicity, and a lot of positive feedback, gave me the trust and self-confidence to continue working on MagicGrid.

After working with multiple customers from various industries, I do agree that there is no single unified approach to this issue, and there cannot be one. Every industry and every customer has their own specifics, and we have taken this into account. MagicGrid has evolved as a foundation and collection of best practices that can be modified or extended to support specific customer needs. Know-how captured in this approach is highly influenced by the INCOSE handbook, ISO/IEC/IEEE 15288, the Unified Architecture Framework (previously known as UPDM) and development efforts. Companies which influenced our efforts included Bombardier Transportation, Kongsberg Defense and Airspace, Ford Motor Company, John Deere, BAE Systems, and many others which I personally, along with fellow No Magic team members and colleagues, have been working with in recent years.

I'm proud that besides the MagicGrid framework, we have managed to put together a detailed step-bystep tutorial and publish it both as a book and an e-book. Please note that this book can only get better with your help. Please do not hesitate to contact us and provide your feedback to make the second edition of the book even better.

My hope is that this book will be a lighthouse for all MBSE practitioners sailing in the dark, and a light breeze for those sailing in the sunlight.

— Aurelijus Morkevicius, Ph.D.
Author of the Idea
Head of Solutions, No Magic, Inc.

Contributors

Andrius Armonas, Ph.D. andrius.armonas@nomagic.com
Barry Papke, barry.papke@nomagic.com
Donatas Mazeika, donatas.mazeika@nomagic.com
Nerijus Jankevicius, nerijus@nomagic.com
Rokas Bartkevicius, rokas.bartkevicius@nomagic.com
Saulius Pavalkis, Ph.D. saulius.pavalkis@nomagic.com
Zilvinas Strolia, zilvinas.strolia@nomagic.com

Reviewers

Antonio Tulelli
Arnaud Durantin
Atif Mahboob
Bernhard Meyer
Bryan K. Pflug
Carmelo Tommasi
Christian Konrad
Darius Silingas
David D. Walden
Gauthier Fanmuy
Himanshu Upadhyay
Holger Gamradt
James Towers
Jonathan Jansen
Macaulay Osaisai
Oliver Nagel
Olivier Casse
Paolo Neri
Pierfelice Ciancia
Raymond Kempkens
Sagar Nirgudkar
Sven Degenkolbe
Thomas Eickhoff
Timo Wekerle
Toshihiro Obata
Ulf Koenemann

Preface

Why this book?

Any theory needs practice. An approach becomes useful when it is provided in company with comprehensive instructions on how to use it in practice. The MagicGrid approach for MBSE by No Magic is no exception to this rule. This was the main reason for writing the book.

So this book is for you, if you're new at MBSE or seek to learn a new MBSE approach. This book is for you, if you wish to learn how to use No Magic software for MBSE in combination with SysML and the new MBSE approach. This book is for you, if you go for comprehensive instructions, illustrated by the case study of modeling an easy-to-understand real-world system.

The book is easier to read if you are familiar with UML/SysML and basic concepts of No Magic software for MBSE; that is, Cameo Systems Modeler or MagicDraw with the SysML plugin installed on the top, generally referred to as the modeling tool.

Why MagicGrid?

The idea of developing a new method for MBSE emerged while working with organizations from various industry sectors, such as defense, automotive, aerospace, and healthcare, to name a few. They needed an unambiguous approach for developing system models using SysML, the de facto language for MBSE, as defined by the International Council on Systems Engineering (INCOSE). However, quite a few methods for MBSE were available in the market at that time, and the existing ones were too abstract for solving a real-world problem. The practice also revealed that every industry and every customer has their own specifics, and cannot use the standard method without applying any modifications or extensions to it.

The MagicGrid approach has evolved by summarizing the experience of numerous MBSE adoption projects, as a foundation and collection of best practices that can be modified or extended to support specific customer needs.

The approach is completely applicable in practice for the following reasons:

- It is fully compatible with "vanilla" SysML. No extensions for SysML are required.
- It clearly defines the modeling process, which is based on the best practices of the systems engineering process.
- It is tool-independent, as long as that tool supports SysML.

MagicGrid 101

As the title reveals, the MagicGrid approach is based on the framework, which can be represented as a

Zachman style matrix (see the following table), and is designed to guide the engineers through the modeling process and answer their questions, like "how to organize the model?", "what is the modeling workflow?", "what model artifacts should be produced in each step of that workflow?", "how these artifacts are linked together?", and so on.

As you can see, the approach includes the definition of the problem, solution, and implementation domains in the system model. They align with the processes defined by ISO/IEC/IEEE 15288 as follows: problem domain with the Stakeholder Needs Development process, solution domain with the Architecture Definition process, and implementation domain with the Design Definition process. Each domain is represented as a separate row of the MagicGrid framework. The row that represents the problem domain splits into two, to convey that the problem domain should be defined by looking at the system of interest (SoI) from the black-box and then from the white-box perspective. The row that represents the solution domain splits into numerous inner rows to convey that the solution architecture of the system can be specified in multiple levels of detail. The row that represents the implementation domain doesn't split and is not fully highlighted like the upper rows, to convey that everything except the requirements specification in this domain is not a part of MBSE and therefore appears outside the scope of the MagicGrid approach.

Each domain definition includes four different aspects of the system to be considered and captured in the model. These aspects match the four pillars of the SysML, that is, requirements, behavior, structure, and parameters (also known as parametrics). They are represented as columns of the matrix.

	PILLAR						
DOMAIN			**Requirements**	**Behavior**	**Structure**	**Parameters**	**Specialty Engineering** **Integrated Testing** **Analysis**
	Problem	**Black Box**	B1-W1 Stakeholder Needs	B2 Use Cases	B3 System Context	B4 Measurements of Effectiveness	
		White Box		W2 Functional Analysis	W3 Logical Subsystems Communication	W4 MoEs for Subsystems	
	Solution		S1 System Requirements	S2 System Behavior	S3 System Structure	S4 System Parameters	
			SS1 Subsystem Requirements	SS2 Subsystem Behavior	SS3 Subsystem Structure	SS4 Subsystem Parameters	
			...	...	...	...	
			C1 Component Requirements	C2 Component Behavior	C3 Component Structure	C4 Component Parameters	
	Implementation		I1 Physical Requirements	Software, Electrical, Mechanical			

A cell at the intersection of some row and column represents a view of the system model, which can consist of one or more presentation artifacts. The presentation artifact can be a diagram, matrix, map, or table.

Though it is not conveyed in the framework, more than one solution architecture can be provided for a single problem. In such case, a trade-off analysis is performed to choose an optimal solution for implementation. Please note that the trade-off analysis, together with specialty engineering and integrated testing, is not covered in this book. However, these are possible extensions of the MagicGrid approach, to be included into future editions of this book.

Case study

In this book, the modeling approach applies to the specification and design of the Vehicle Climate Control System (VCCS). Technical accuracy and the feasibility of the actual solution proposed are not high priorities.

How to read this book

While reading the book, it's important to understand that it doesn't replace the modeling tool documentation. It is intended to supplement it.

The structure of the book corresponds the layout of the MagicGrid framework. It is divided into three main chapters, each describing the modeling workflow within a single domain. These are the Problem domain (including the Black-box perspective and White-box perspective sub-chapters), the Solution domain, and the Implementation domain. You should read them in succession.

Each chapter consists of multiple sections. Sections are named after the cells from relevant domain. The order of these sections defines the modeling workflow in that domain. Some sections are named with suffixes, such as "initial" and "final", which means that the modeling in the relevant cell is divided into two phases, and the presentation artifacts of that cell are delivered in the final phase; for example, B3. System Context: initial and B3. System Context: final with B2. Use Cases in-between. Every section includes a brief overview of the cell by explaining the purpose of modeling in this cell, who is responsible for the modeling, how to model, and what the subsequent cells are.

Every section (except for a few) consists of multiple sub-sections, which provide comprehensive instructions of how to model in the related cell. Therefore, they all have prefixes with the pattern "Step X", where "X" stands for the step sequence number. A sub-section provides answers to the questions, like "what to do?", "why to do it?", and "how to do it?". Follow these instructions from the very beginning until the end of the book to create a sample model of the VCCS by yourself.

Some sub-sections contain information boxes marked with the ⓘ symbol. These boxes are designated to provide you with additional information, tips and tricks for smarter modeling, or references to more detailed information sources.

All of the keywords and acronyms are defined in the Glossary.

How to provide feedback

We appreciate any feedback from our readers for the next version of this book. Send us an email at magicgrid@nomagic.com.

Problem domain

Introduction

The purpose of the problem domain definition is to analyze stakeholder needs and refine them with SysML model elements to get a clear and coherent description of what problems the SoI must solve. As you can see in the following table, the problem domain analysis is performed in two phases. At the first phase, the SoI is considered a black box and the main focus is on how it interacts with the environment without getting any knowledge about its internal structure and behavior. In other words, the operational analysis of the SoI is performed. At the second phase, the black box is opened and the SoI is analyzed from the white-box perspective, which helps to understand in detail how the system shall operate. In other words, the functional analysis of the SoI is performed. It is also important to note that the problem definition doesn't cover the physical architecture of the SoI.

As you can see in the following table, both phases of the problem domain definition consist of analyzing the requirements, behavior, structure, and parameters of the SoI. The only difference is in perspective.

Modeling in the problem domain is described step by step in related pages.

	PILLAR						
DOMAIN			**Requirements**	**Behavior**	**Structure**	**Parameters**	**Specialty Engineering** **Integrated Testing** **Analysis**
	Problem	**Black Box**	B1-W1 Stakeholder Needs	B2 Use Cases	B3 System Context	B4 Measurements of Effectiveness	
		White Box		W2 Functional Analysis	W3 Logical Subsystems Communication	W4 MoEs for Subsystems	
	Solution		S1 System Requirements	S2 System Behavior	S3 System Structure	S4 System Parameters	
			SS1 Subsystem Requirements	SS2 Subsystem Behavior	SS3 Subsystem Structure	SS4 Subsystem Parameters	
			...	...	...	...	
			C1 Component Requirements	C2 Component Behavior	C3 Component Structure	C4 Component Parameters	
	Implementation		I1 Physical Requirements	Software, Electrical, Mechanical			

Black-box perspective

Introduction

Problem domain analysis and definition from the black-box perspective comprise the first step towards identifying and specifying the problem the system should solve. In this phase of problem domain definition, the SoI is considered a black box, which means that only inputs and outputs of the system are analyzed, without getting any knowledge about its internal structure and behavior. Interactions between the SoI, other systems, and supposed users of the SoI in a variety of system contexts are analyzed instead.

Modeling the problem domain from the black-box perspective is explained step by step in related pages.

	PILLAR						
DOMAIN			**Requirements**	**Behavior**	**Structure**	**Parameters**	
	Problem	**Black Box**	B1-W1 Stakeholder Needs	B2 Use Cases	B3 System Context	B4 Measurements of Effectiveness	**Specialty Engineering**
		White Box		W2 Functional Analysis	W3 Logical Subsystems Communication	W4 MoEs for Subsystems	
	Solution		S1 System Requirements	S2 System Behavior	S3 System Structure	S4 System Parameters	**Integrated Testing**
			SS1 Subsystem Requirements	SS2 Subsystem Behavior	SS3 Subsystem Structure	SS4 Subsystem Parameters	
			...	...	...	...	**Analysis**
			C1 Component Requirements	C2 Component Behavior	C3 Component Structure	C4 Component Parameters	
	Implementation		I1 Physical Requirements	Software, Electrical, Mechanical			

B1-W1. Stakeholder Needs: initial

What is it?

The cell represents information gathered from various stakeholders of the system. This information includes primary user needs, system-related government regulations, policies, procedures, and internal guidelines, to name a few.

In the initial phase of this cell, stakeholder needs are captured. This can be done by interviewing the stakeholders, giving them questionnaires, discussing needs in focus groups, or studying documents written in diverse formats. Though elicited information is raw, it does not need to be specially rewritten. However, it needs to be analyzed and refined in other cells of the problem domain model. These refinements afterwards serve as a basis for specifying system requirements (see S1.initial). Thus, it can be stated that system requirements are derived from stakeholder needs.

In order to have a complete problem domain model, you need to perform the tasks determined by the final phase of this cell. That is, you need to establish traceability relationships between stakeholder needs and the rest of the problem domain model to convey how behavioral and structural elements of the SoI refine stakeholder needs.

Who is responsible?

In a typical multidisciplinary systems engineering team, elicitation and analysis of stakeholder needs are performed by the *Requirements Team*.

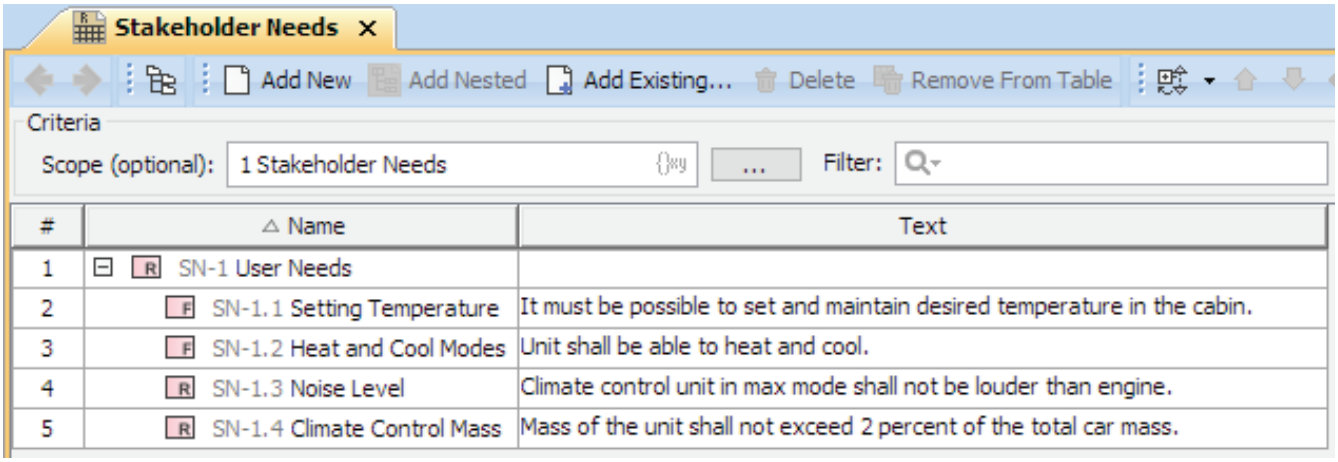
How to model?

Stakeholder needs can be captured directly in the modeling tool, or alternatively, they can be imported from other tools and formats, such as:

- Requirements management tools (for example, IBM® Rational® DOORS®)
- Spreadsheets (for example, Microsoft Excel file)
- ReqIF file (which contains data exported from another tool, that is, IBM® Rational® DOORS®, PTC Integrity Modeler, Polarion® REQUIREMENTS™, or Siemens Teamcenter)

In the modeling tool, a single stakeholder need can be captured as a SysML requirement, which has a unique identification, name, and textual specification. The entire list of stakeholder needs can be displayed in a SysML requirement table

(see the following figure) or diagram. Stakeholder needs can be hierarchically related to each other (grouped) by the containment relationships.



#	Name	Text
1	SN-1 User Needs	
2	SN-1.1 Setting Temperature	It must be possible to set and maintain desired temperature in the cabin.
3	SN-1.2 Heat and Cool Modes	Unit shall be able to heat and cool.
4	SN-1.3 Noise Level	Climate control unit in max mode shall not be louder than engine.
5	SN-1.4 Climate Control Mass	Mass of the unit shall not exceed 2 percent of the total car mass.

What's next?

- Stakeholder needs are used to define system contexts in B3.initial.
- Functional stakeholder needs are further analyzed and refined by use cases and use case scenarios in B2.
- Non-functional stakeholder needs are refined by measurements of effectiveness (value properties) in B4.

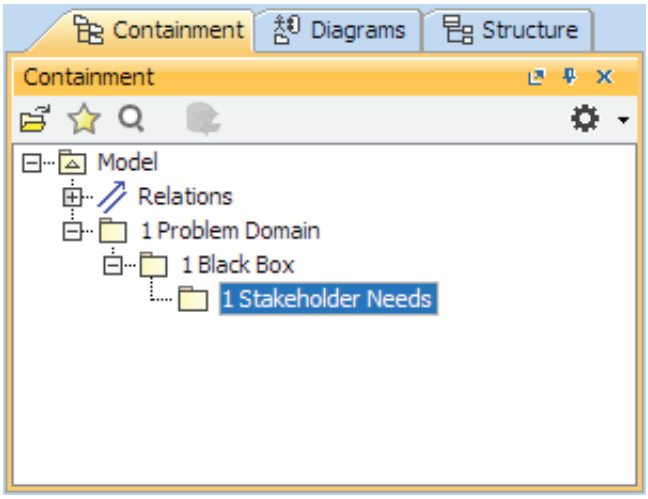
Tutorial

Step 1. Organizing the model for B1-W1

A well-organized model is easier to read, understand, and maintain. That's why we recommend organizing your model into packages. Packages in a SysML model are very similar to folders in a file system Graphical User Interface (GUI): while folders are used to organize files, packages are used to organize diagrams and elements (including other packages).

According to the design of the MagicGrid framework, model artifacts that capture stakeholder needs should be stored under the structure of packages displayed in the following figure. As you can see, the top-level package represents the domain, the medium-level package represents the perspective, and the bottom-level package represents the cell.

ⓘ You can skip this step (as well as other steps of establishing the model structure), if you use the *MagicGrid Quick-Start* or *MagicGrid Blank* template. They both come with the 19.0 version of the modeling tool, and include a predefined structure based on the MagicGrid framework.



ⓘ Using numbers in package names enables you to preserve the package location in the model tree.

To organize the model for B1-W1

1. Right-click the *Model* package (this is the default name of the root package) and select **Create Element**.
2. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.
3. Type *1 Problem Domain* to specify the name of the new package and press Enter.
4. Right-click the *1 Problem Domain* package and select **Create Element**.
5. Repeat steps 2 and 3 to create the package named *1 Black Box*.
6. Right-click the *1 Black Box* package and select **Create Element**.
7. Repeat steps 2 and 3 to create the package named *1 Stakeholder Needs*.

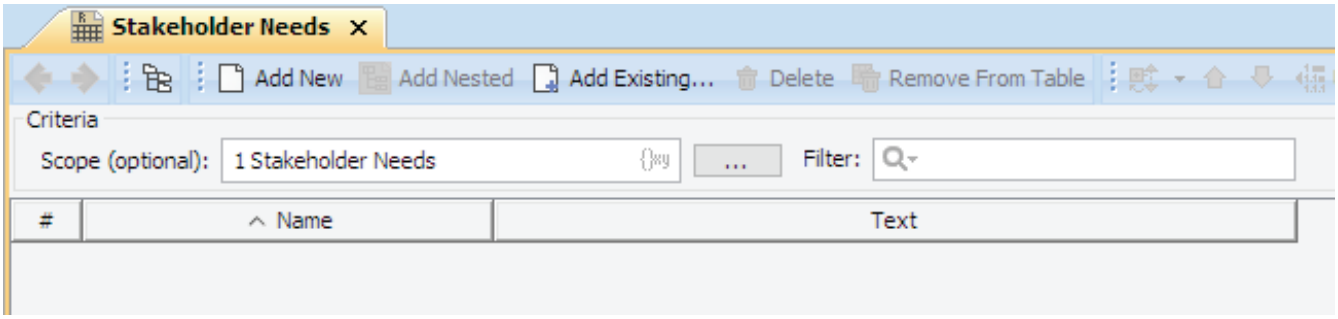
Step 2. Creating a table for stakeholder needs

For capturing and displaying stakeholder needs, we recommend utilizing the SysML requirement table.

To create the SysML requirement table for capturing stakeholder needs

1. Right-click the *1 Stakeholder Needs* package and select **Create Diagram**.
2. In the search box, type *rt*, where r stands for *requirement* and *t* for *table*, and then double-press Enter. The table is created.

   ⓘ Note that the table is named after the package where is it stored. This name suits for the table, too. You need only remove the sequence number from its name.
3. Select the *1 Stakeholder Needs* package and drag it to the **Scope** box in the **Criteria** area of the table. From now on, the table is updated every time you append the contents of the package *1 Stakeholder Needs.*



Step 3. Capturing stakeholder needs

Let's say you need to capture the following stakeholder needs into your model:

#	Name	Text
1	Setting Temperature	It must be possible to set and maintain desired temperature in the cabin.
2	Heat and Cool Modes	Unit shall be able to heat and cool.
3	Noise Level	Climate control unit in max mode shall not be louder than the engine.
4	Climate Control Mass	Mass of the unit shall not exceed 2 percent of the total car mass.

A single stakeholder need can be stored as a SysML requirement, which has an unique identification, name, and text specification.

You can get the stakeholder needs into your model by one of the following means:

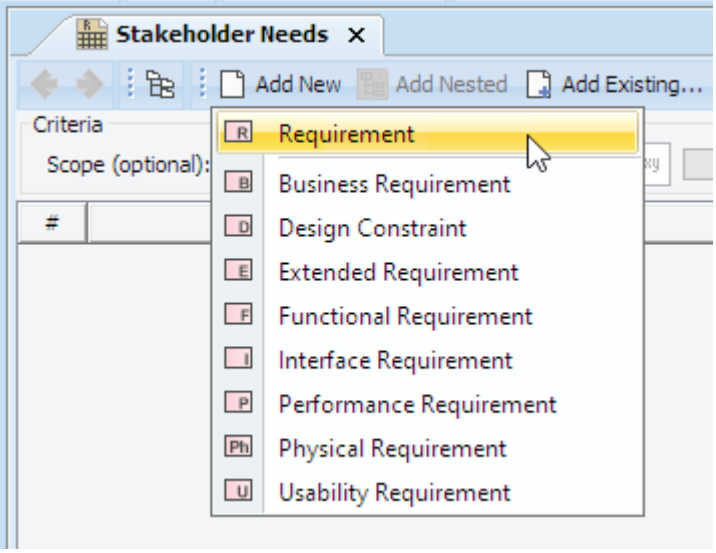
- Capturing information directly in the model

- Copying information from a Microsoft Excel spreadsheet
- Importing information from a ReqIF file
- Synchronizing information from IBM Rational DOORS

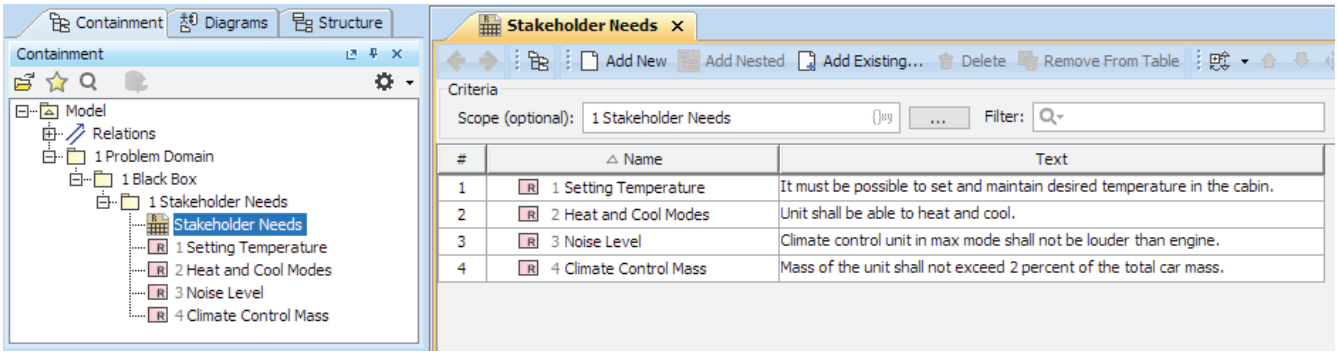
Capturing information directly in the model

To capture stakeholder needs directly in the model

1. Open the table created in step 2 of the B1-W1 initial phase tutorial, if not yet opened.
2. On the toolbar of the table, click **Add New** and select **Requirement**.



3. In the newly created row, type the name and text of the first stakeholder need displayed in the table within step 3 of the B1-W1 initial phase tutorial.
4. Repeat steps 2 and 3 until you have all the items in the table as displayed in the following figure



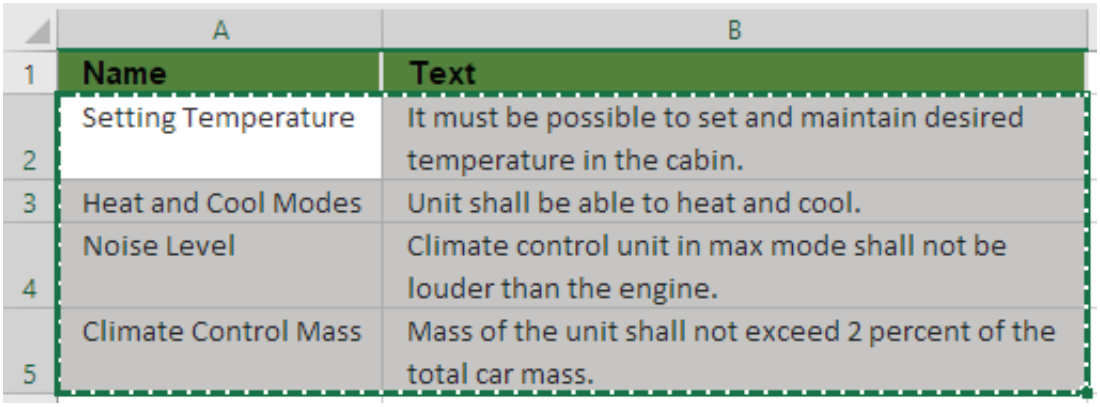
ⓘ Items created in the SysML requirements table appear in the Model Browser, too.

Copying information from a Microsoft Excel spreadsheet

If you're provided with the list of stakeholder needs in a Microsoft (MS) Excel spreadsheet, you can easily transfer this information to the SysML model by utilizing the copy-paste capability of the modeling tool.

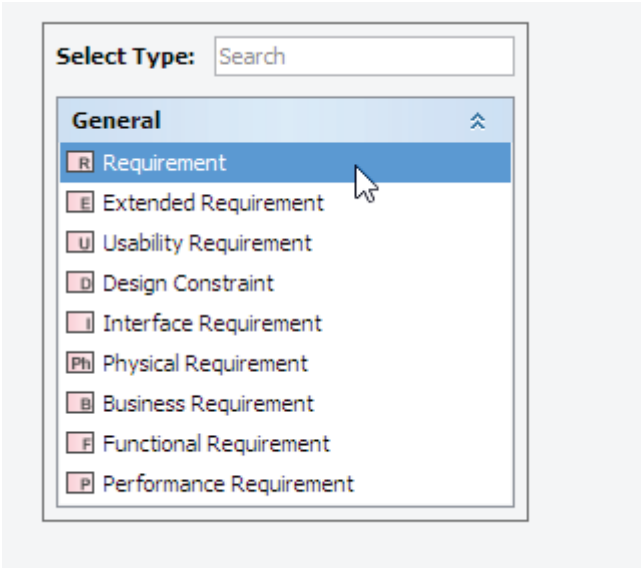
To copy stakeholder needs from the MS Excel spreadsheet

1. Open the spreadsheet and select the contents you want to copy to your model.

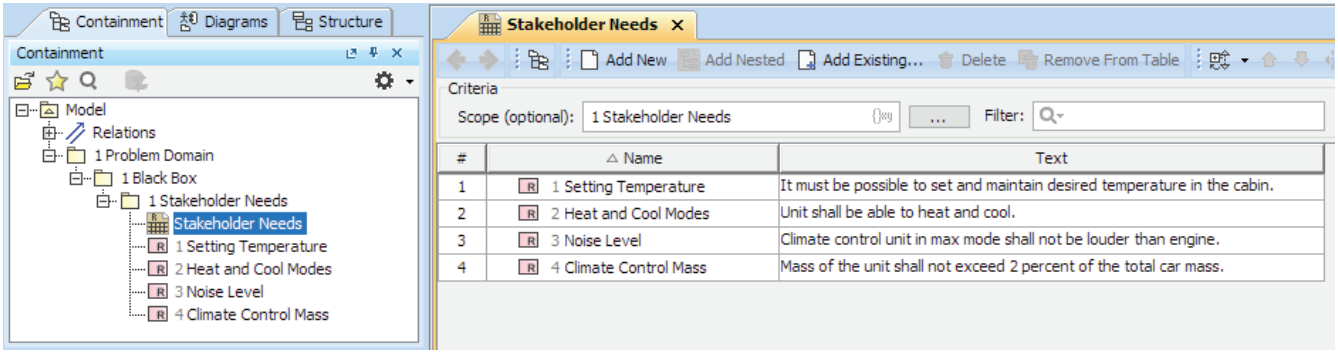


	A	B
1	**Name**	**Text**
2	Setting Temperature	It must be possible to set and maintain desired temperature in the cabin.
3	Heat and Cool Modes	Unit shall be able to heat and cool.
4	Noise Level	Climate control unit in max mode shall not be louder than the engine.
5	Climate Control Mass	Mass of the unit shall not exceed 2 percent of the total car mass.

2. Copy the contents:
   - Press Ctrl+C on Windows or Linux.
   - Press Command+C on macOS.
3. Switch to the modeling tool and open the SysML requirement table created in step 2 of the B1-W1 initial phase tutorial, if not yet opened.
4. Paste the contents to the table:
   - Press Ctrl+V on Windows or Linux.
   - Press Command+V on macOS.
5. Select **Requirement** as the type for the pasted data.



6. Wait while stakeholder needs are pasted into the table.



ⓘ Items copied to the SysML requirements table appear in the Model Browser, too.

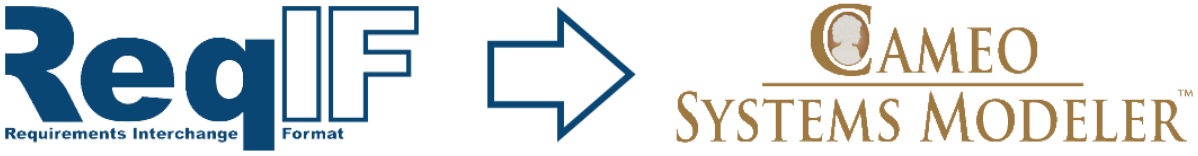
Importing information from a ReqIF file

It may happen that stakeholder needs are gathered by using one of these tools:

- IBM® Rational® DOORS®
- PTC Integrity Modeler
- Polarion® REQUIREMENTS™
- Siemens Teamcenter
- Dassault Systèmes Reqtify

In such case, information between a requirements management tool and the modeling tool can be transferred via Requirements Interchange Format (ReqIF) file. Thus, getting stakeholder needs into your model includes:

1. Exporting requirements from the requirements management tool to a ReqIF file.
2. Importing the ReqIF file to the SysML model in the modeling tool.

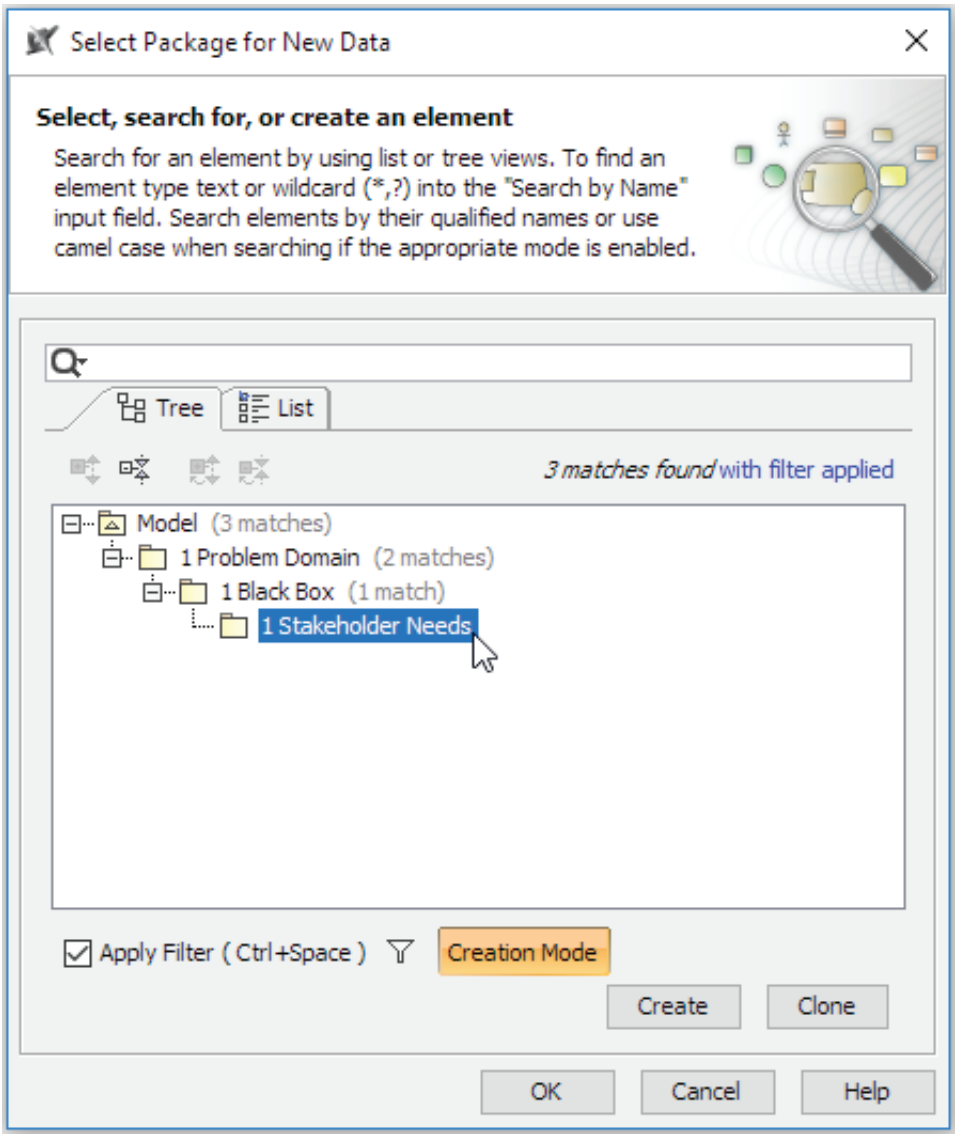


We skip the first step, assuming that you've already been provided with the ReqIF file.

To import stakeholder needs from the ReqIF file

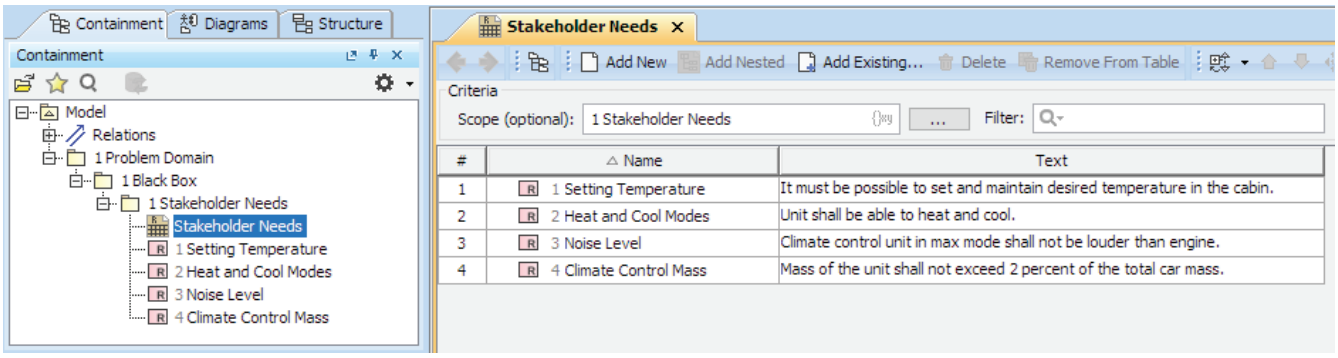
1. From the main menu, select **File** > **Import From** > **Requirements Interchange Format (ReqIF) File**.
2. Select the ReqIF file form your file system and click **Open**.

3. In the **Select Package for New Data** dialog, select the package wherein you want to store the imported stakeholder needs; that is, the package *1 Stakeholder Needs*.



4. Click **OK**.

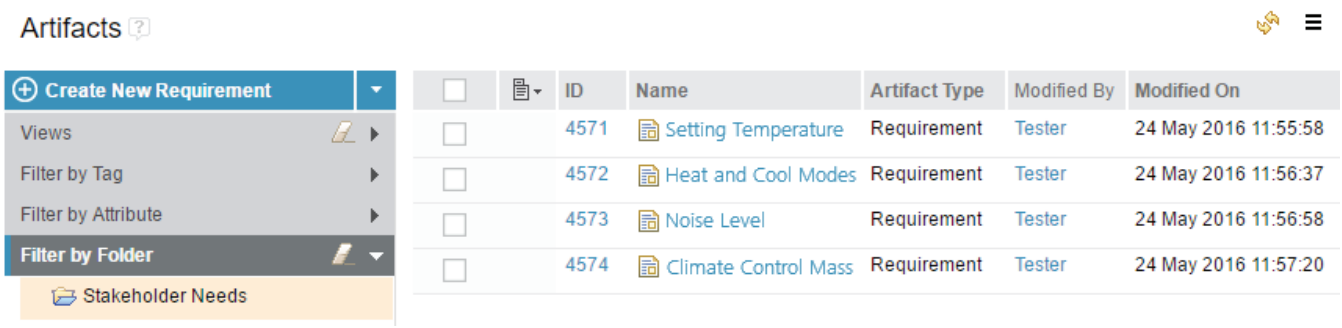
5. Wait while stakeholder needs are imported into the *1 Stakeholder Needs* package. Since that package is the scope of the SysML requirement table (see step 2 of the B1-W1 initial phase tutorial), all imported items appear in the table too.



Synchronizing information from IBM Rational DOORS

If stakeholder needs are gathered in an IBM® Rational® DOORS® project (see the following figure), you can get them into the SysML model by synchronizing data between DOORS and the modeling tool.

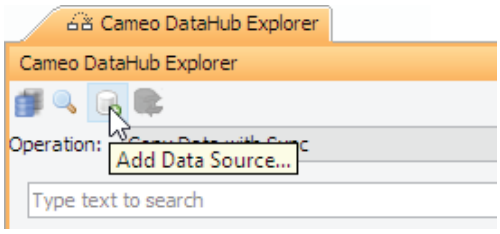
> ⓘ The modeling tool can be synchronized with DOORS only if it has the Cameo DataHub Plugin installed.



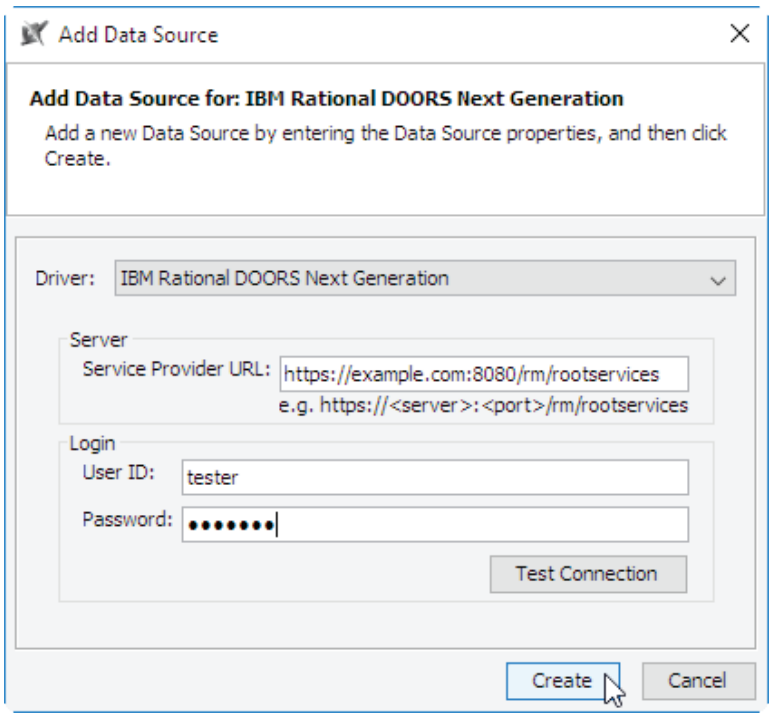
In this tutorial, we assume that the *Requirements Team* uses DOORS only for capturing stakeholder needs, and further analysis is performed in the SysML model. For this reason, we need to establish oneway synchronization between DOORS and the modeling tool. In one-way synchronization, items are copied from DOORS to the modeling tool, but no items are ever copied back from the modeling tool to DOORS.

To synchronize stakeholder needs from DOORS NG 6.x to the modeling tool

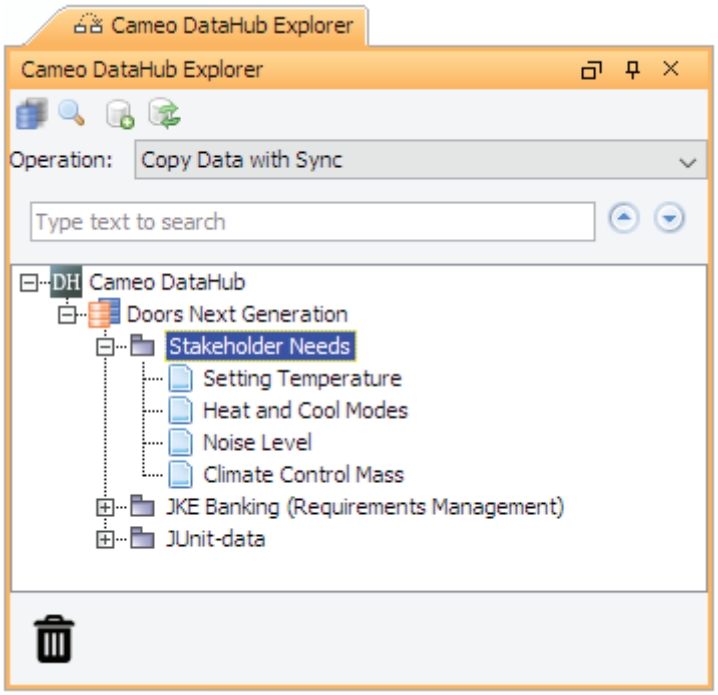
1. Be sure you have Cameo DataHub Plugin 18.1 or later installed on top of the modeling tool.
2. Connect to the DOORS NG 6.x repository:
   a. From the main menu of the modeling tool, select **Tool** > **DataHub** > **DataHub Explorer.** The **Cameo DataHub Explorer** panel opens on the right of the modeling tool window.
   b. On the toolbar of the open panel, click the Add Data Source button.



c. Provide the required information and click **Create**, as shown in the following figure.



d. Wait while the modeling tool connects to the DOORS NG 6.x repository. After the operation is completed, the tree of the **Cameo DataHub Explorer** panel displays projects from the DOORS repository (as packages). One of the projects is *CCU Stakeholder Needs*.



3. Synchronize data:

a. On the toolbar of the **Cameo DataHub Explorer** panel, click the **Operation** drop-down list box and select **Copy Data with Sync**.

b. In the **Cameo DataHub Explorer** tree, select the *CCU Stakeholder Needs* package and drag it onto the *1 Black Box* package in the Model Browser (on the left of the modeling tool window).

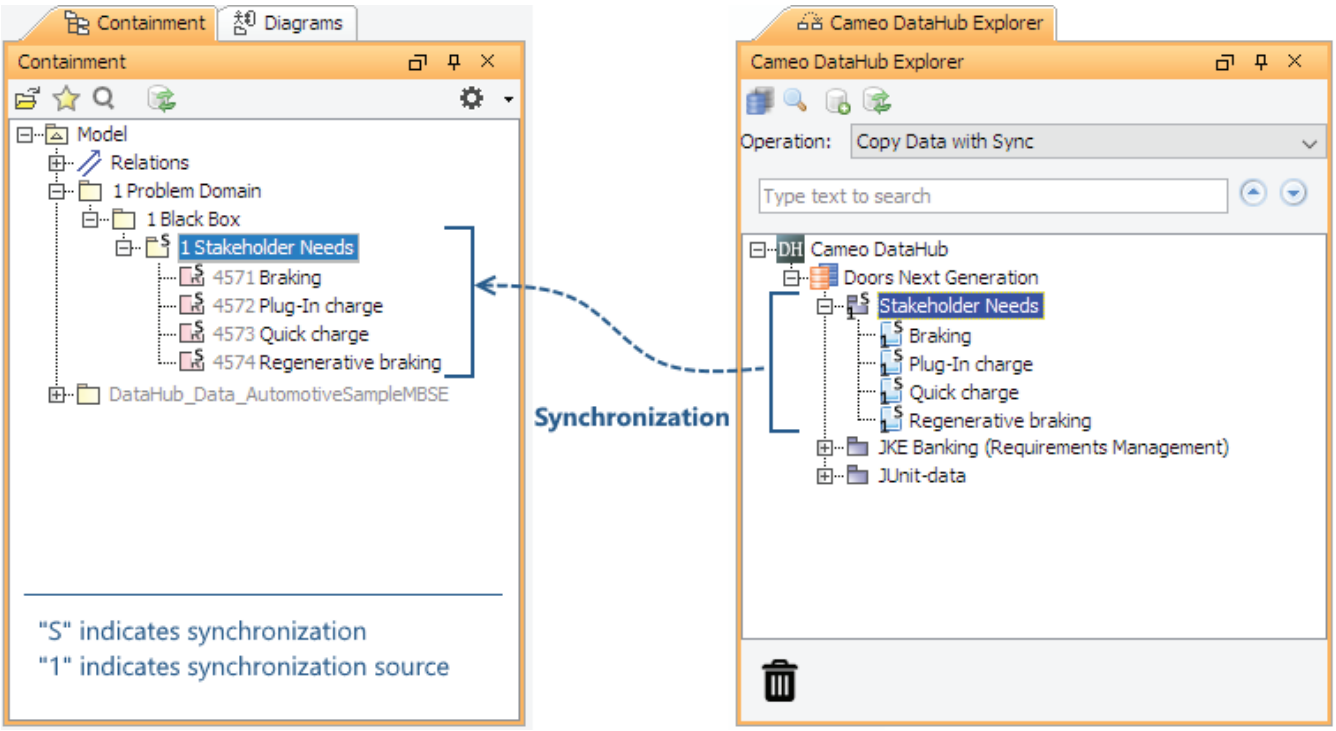
> ⓘ In this case, you should ignore the inner package *1 Stakeholder Needs* created in step 1 of the B1-W1 initial phase tutorial. If you decide to ignore this recommendation and drag the items of stakeholder needs to the package *1 Stakeholder Needs* one by one (ignoring the package *CCU Stakeholder Needs*), keep in mind that you are taking the risk of missing the items that will be created in the DOORS project afterwards. In such case, only individual items are synchronized, not the whole project.

c. Wait for the message announcing that synchronization is completed. Synchronized items in both trees become marked with *S* (see the following figure).

4. Make the synchronization single directional:

a. In the Model Browser of the modeling tool, right-click the *CCU Stakeholder Needs* package and select **DataHub Actions** > **DHLink Panel.** The **DHLinks** panel opens at the bottom of the modeling tool window.

b. Right-click the **Direction** cell and select Change **Direction** > **Sync to MagicDraw**. The synchronization is changed to single directional, and source items become marked with *1*.



Step 4. Grouping stakeholder needs

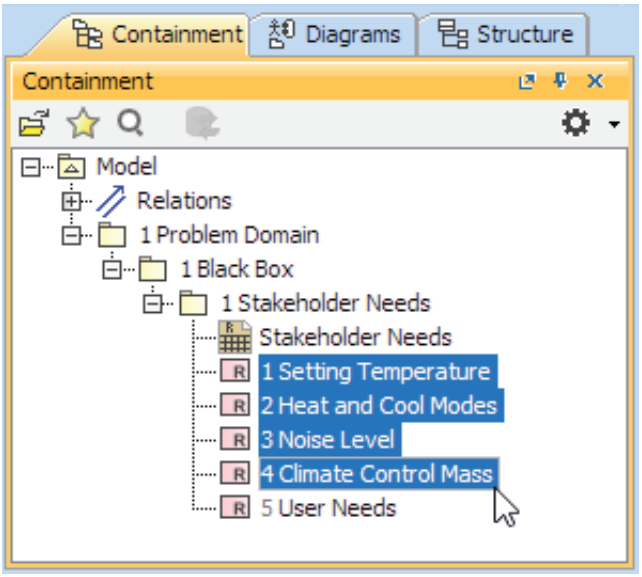
After you have stakeholder needs in your model, you can arrange them into groups according to their nature. These groups can be system-related government regulations, user needs, or business requirements, to name a few. It is quite common that stakeholder needs of different natures are even moved into the model from different sources. For example, user needs can be synchronized from a requirements management tool, and regulations can be copied from the MS Excel spreadsheet.

Now let's analyze the stakeholder needs of the vehicle Climate Control Unit in your model. Actually, they are user needs, and thus can be grouped under the element *User Needs* of the SysML requirement type. We recommend using a regular SysML requirement as a grouping element. The grouping element might not have any text itself.

To group stakeholder needs

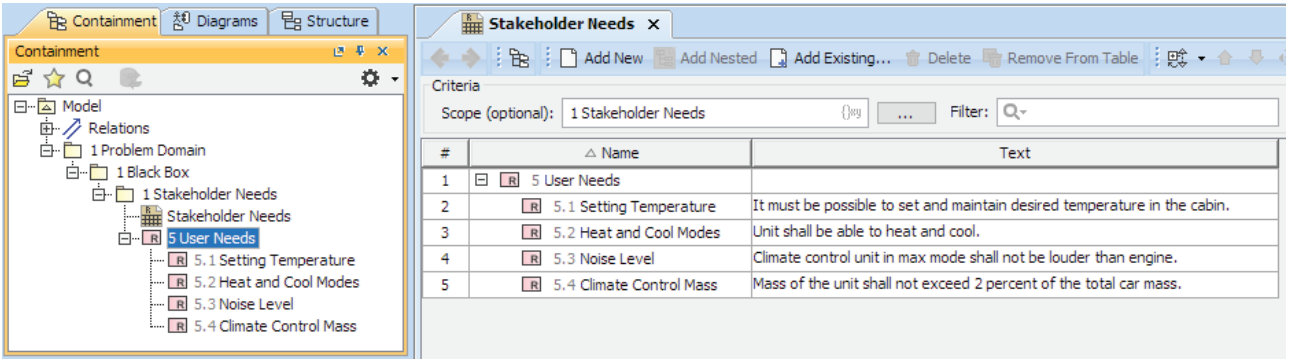
1. Create a grouping requirement:
   a. In the Model Browser, right-click the *1 Stakeholder Needs* package and select **Create Element.**
   b. In the search box, type *re* – the first letters of the requirement type, and press Enter. A new element of the requirement type is created.
   c. Type *User Needs* to specify the name of the requirement and press Enter.
2. In the Model Browser, select all previously captured stakeholder needs and drag them onto the *User Needs* requirement. The stakeholder needs become nested under the grouping requirement *User Needs*.

> ⓘ To select the set of adjacent items in the tree, click the first one, press Shift, and while holding it down, select the last one.



3. See appropriate changes in the SysML requirement table.

> ⓘ Be sure the table is displayed in the Compact tree mode. For this, click the Options button on the table toolbar and then select **Display Mode** > **Compact Tree**.



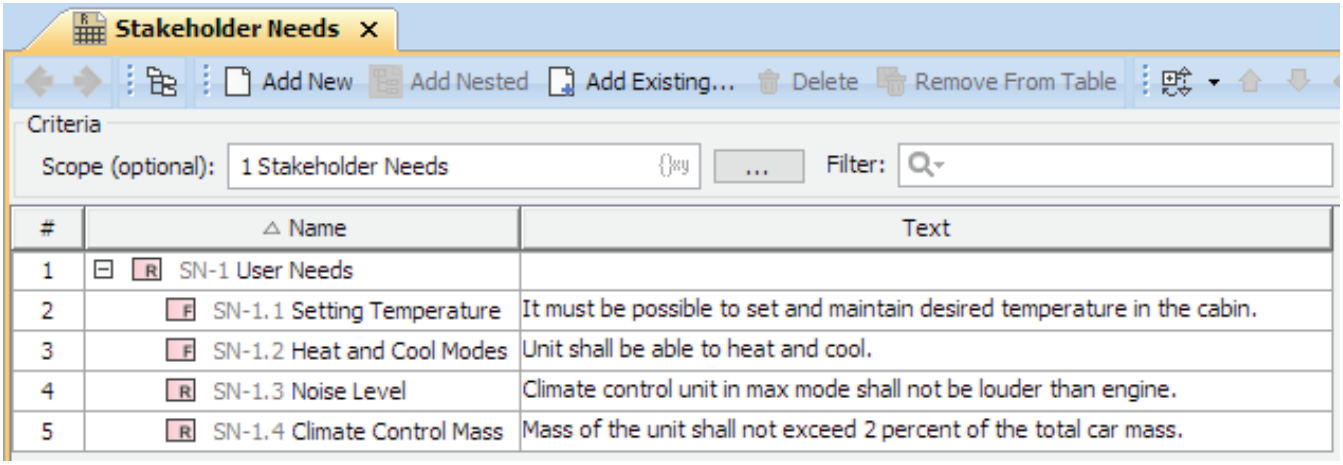
Step 5. Numbering stakeholder needs

SysML requirements, which we use for capturing and storing stakeholder needs, are by default numbered using integers. We recommend adding prefixes to these simple numbers in order to distinguish stakeholder needs from other requirements, such as system, component, and physical requirements, to be created later on. Let's say we want to add a prefix *SN-* to these numbers (*S* stands for *stakeholder* and *N* for *needs*).

We also need to reset numbering from *1*, since the initial number has automatically changed to *5* after grouping.

To number stakeholder needs with the prefix ***SN-***

1. In the Model Browser, right-click the grouping requirement *User Needs* and select **Element Numbering**. The **Element Numbering** dialog opens.
2. In the dialog, click the empty value cell of the **Prefix** column to initiate the editing, type *SN-* and press Enter. Every item of stakeholder needs in the model tree within the dialog becomes numbered with the prefix *SN-*.
3. Click **Details** to see more options.
4. Click to select the **Renumber from Initial Value** check box.
5. Click the **Renumber Recursively** button.
6. Click **OK** to close the dialog.



#	Name	Text
1	SN-1 User Needs	
2	SN-1.1 Setting Temperature	It must be possible to set and maintain desired temperature in the cabin.
3	SN-1.2 Heat and Cool Modes	Unit shall be able to heat and cool.
4	SN-1.3 Noise Level	Climate control unit in max mode shall not be louder than engine.
5	SN-1.4 Climate Control Mass	Mass of the unit shall not exceed 2 percent of the total car mass.

Step 6. Categorizing stakeholder needs

Stakeholder needs can be either functional or non-functional. Functional stakeholder needs specify what the SoI is expected to do. They are further refined by use cases and use case scenarios in the B2 cell of the MagicGrid framework. Non-functional stakeholder needs specify non-functional characteristics of the SoI. They are refined by measurements of effectiveness (value properties) in the B4 cell of the framework.

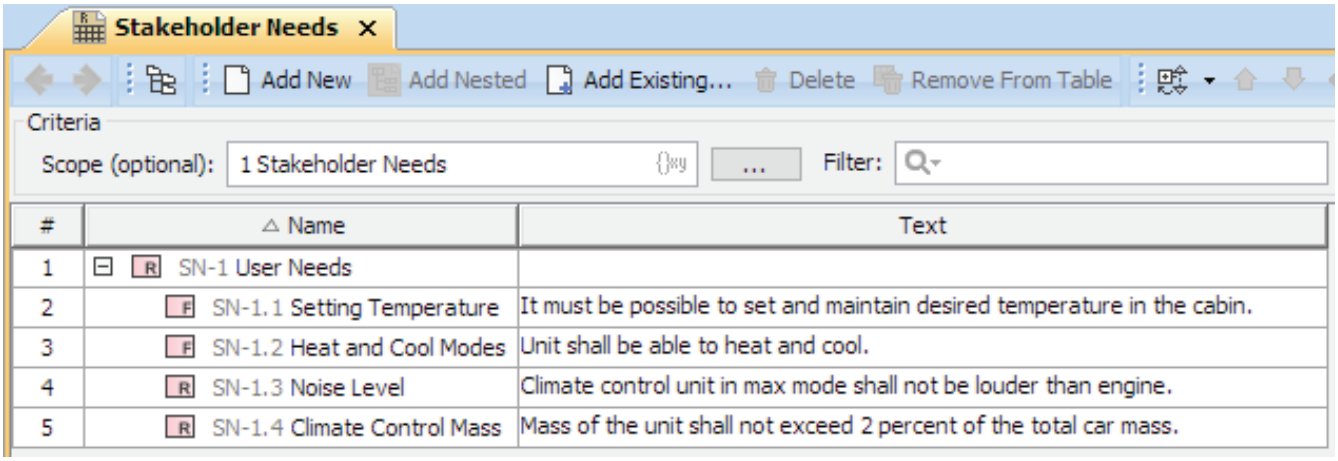
Categorization facilitates further analysis of stakeholder needs, especially when you have to deal with large a scope of information. This step may be skipped, if the case doesn't require it.

Requirements that capture functional stakeholder needs should be converted to functional. Non-functional stakeholder needs can remain simple requirements. It is obvious that both *Setting Temperature* and *Heat and Cool Modes* are functional stakeholder needs.

To convert the ***Setting Temperature*** and ***Heat and Cool Modes*** requirements to functional

1. Right-click both requirements in the Model Browser.

2. Select **Refactor** > **Convert To** > **More Specific** > **Functional Requirement.** Both requirements are converted to functional, and *F* instead of *R* is displayed on each element icon.



#	Name	Text
1	SN-1 User Needs	
2	SN-1.1 Setting Temperature	It must be possible to set and maintain desired temperature in the cabin.
3	SN-1.2 Heat and Cool Modes	Unit shall be able to heat and cool.
4	SN-1.3 Noise Level	Climate control unit in max mode shall not be louder than engine.
5	SN-1.4 Climate Control Mass	Mass of the unit shall not exceed 2 percent of the total car mass.

B3. System Context: initial

What is it?

System context or operating environment determines an external view of the system. It must introduce elements that do not belong to the system but do interact with that system. More than one system context can be defined for a single SoI. Besides the system itself (currently considered to be a black box), the collection of elements in the particular system context can include external systems and users (humans, organizations) that interact with the SoI in that context.

Modeling in this cell includes two phases: initial and final. During the initial phase of B3, one or more system contexts are defined, and elements that participate in each context are identified. In the final phase of B3, interactions within each system context are specified. Since this requires analyzing system behavior, you should identify use cases of that system context and model use case scenarios first. That's why the B3 cell is interrupted by the B2 cell.

The initial phase of the B3 cell produces:

- Definitions of system contexts
- Participants of each system context: SoI, supposed users of t he system, other systems, etc.

Who is responsible?

In a typical multidisciplinary systems engineering team, the system contexts are specified by the *Requirements Team*.

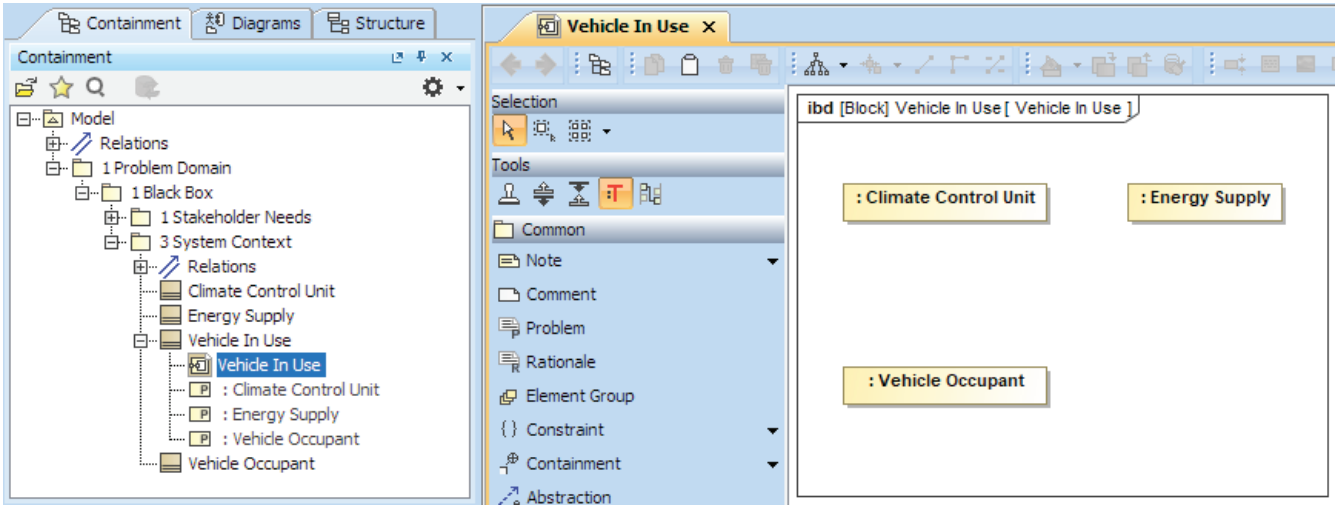
How to model?

In the modeling tool, system contexts can be captured as elements of the SysML block type. Participants of the system context can be specified as part properties of that system context and represented in the SysML internal block diagram

(ibd) created for that system context. Each part is typed by the block, which defines the SoI, the user, or another system in the model.

ⓘ We recommend using blocks rather than actors to define humans (for example, users of the SoI) for the following reasons:

- A block is treated as a part of the system context. An actor is not.
- A block can be marked as external. An actor cannot.
- A block can have its behavior defined. An actor cannot.



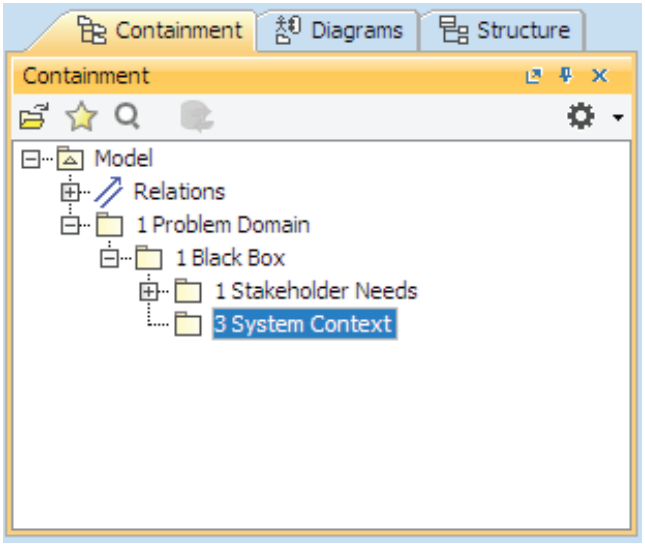
What's next?

- System contexts are used to analyze behavior of the system in B2. Various use cases of the system are grouped into system contexts, each to the one wherein it is performed.
- Participants of each system context captured as part properties of that system context are also used in B2. They can be represented as swimlane partitions in the SysML activity diagram that describes a use case scenario.

Tutorial

Step 1. Organizing the model for B3

Following the structure of the MagicGrid framework, system contexts and owned-by-them SysML internal block diagrams, together with elements they represent, should be stored in a separate package inside the *1 Black Box* package. We recommend naming the package after the cell, that is, *3 System Context*.



To organize the model for B3

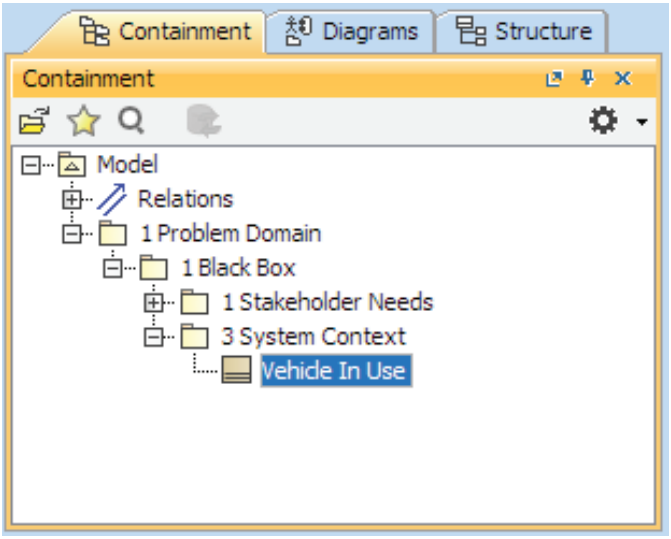
1. Right-click the *1 Black Box* package and select **Create Element**.
2. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.
3. Type *3 System Context* to specify the name of the new package and press Enter.

Step 2. Capturing system contexts

System contexts can be identified by analyzing stakeholder needs captured in B1-W1.initial. Stakeholder needs of the Vehicle Climate Control Unit enable you to identify a single context, the *Vehicle In Use*. It can be captured in your model as an element of the SysML block type.

To create a block that captures the *Vehicle In Use* system context

1. Right-click the *3 System Context* package created in step 1 of the B3 initial phase tutorial and select **Create Element**.
2. In the search box, type *bl*, the initial letters of the required element type, and press Enter.
3. Type *Vehicle In Use* to specify the name of the new block and press Enter. The block is created and represented in the Model Browser.



Step 3. Creating an ibd for a system context

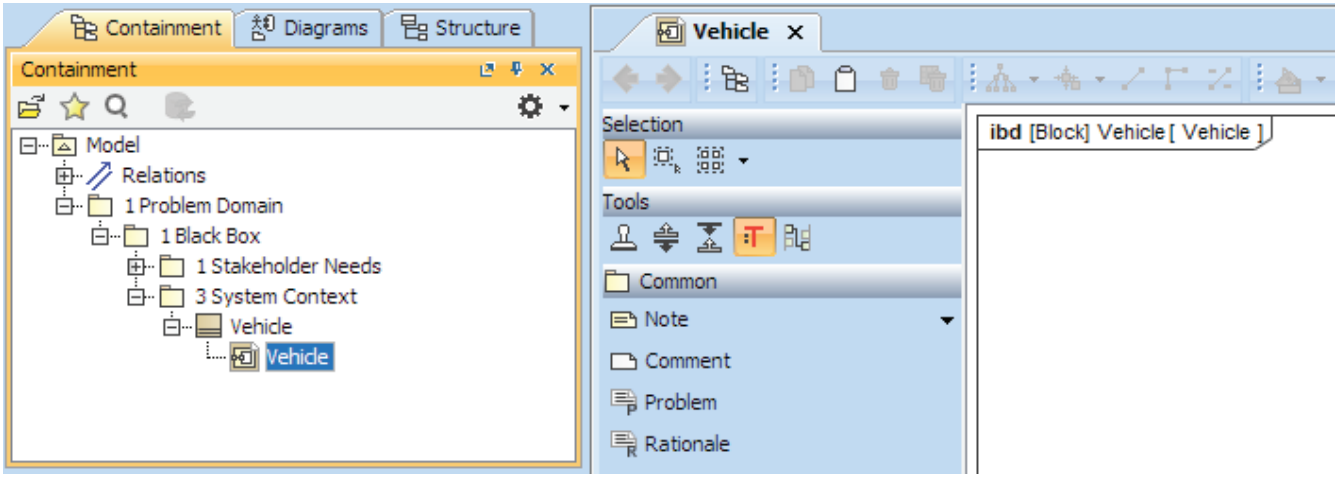
To start specifying participants of the *Vehicle In Use* system context, you first need to create a SysML internal block diagram for the block that captures that system context.

To create an ibd for the *Vehicle In Use* block

1. Right-click the *Vehicle In Use* block and select **Create Diagram**.

2. In the search box, type ibd, the acronym of the SysML internal block diagram, and then doublepress Enter. The diagram is created and represented in the Model Browser under the *Vehicle In Use* block.

> ⓘ The diagram is named after the block that owns it. As this name perfectly suits the diagram too, there is no need to change it.



Step 4. Capturing participants of the system context

Participants of the system context can be captured as part properties, and each part property should be typed by the block that defines the concept of SoI, supposed users of the system (humans, organizations), or some external system in the model.

> ⓘ Here you should remember the SysML Specification, which tells you that blocks should be used for defining concepts, and parts are used for specifying the usage of these concepts. Any concept, once defined in your model, can be used in many system contexts. For example, the Vehicle Climate Control Unit SoI defined as a block can type parts in various system contexts, meaning that it participates within all these contexts.

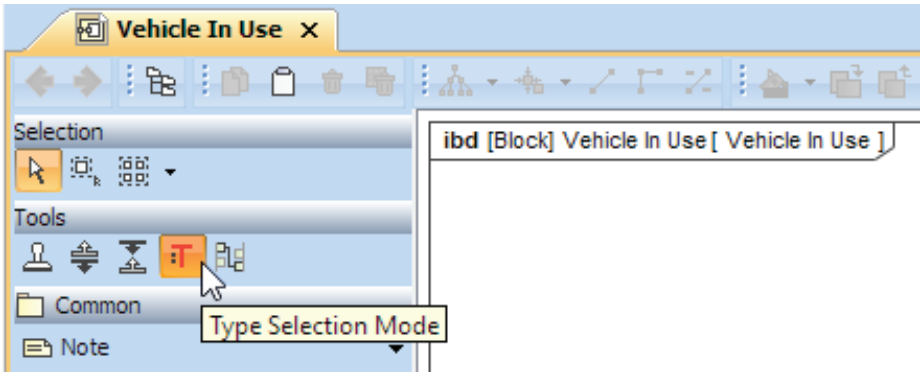
Stakeholder needs indicate that the *Vehicle In Use* system context includes two participants: the Vehicle *Climate Control Unit* and the vehicle occupant (driver or passenger). In addition, the system context must include an external system for supplying the SoI with energy (even though it is not mentioned in stakeholder needs).

To capture participants of the *Vehicle In Use* system context

1. Open the ibd created in step 3 of the B3 initial phase tutorial, if not yet opened.

2. Make sure the Type Selection Mode is on in the diagram. Otherwise, parts created in this diagram will not be typed by blocks.

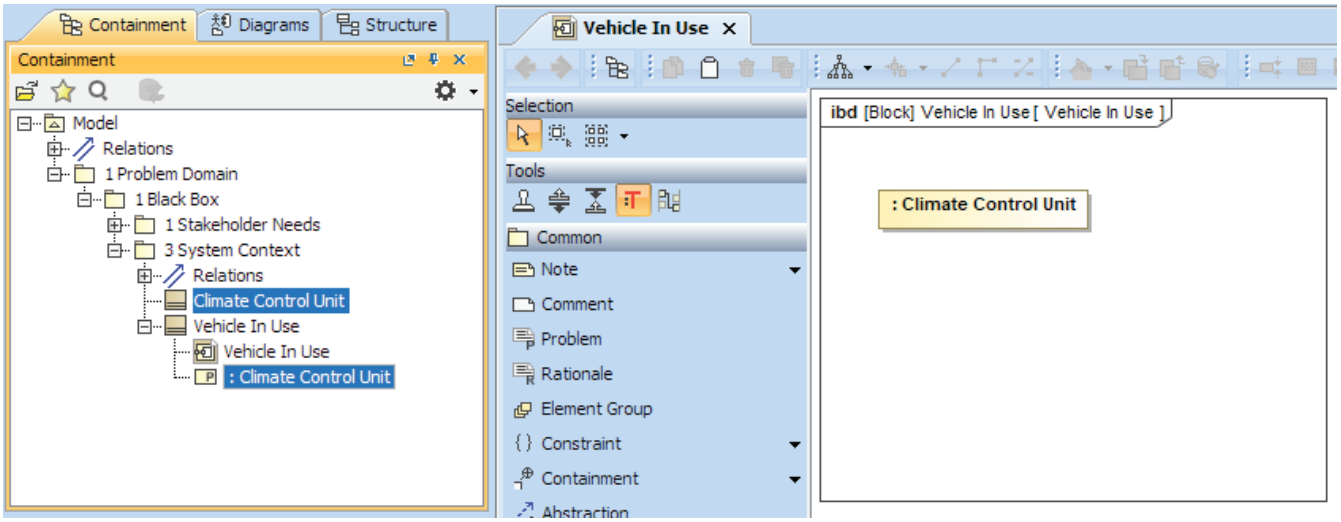
> ⓘ In this mode, when you create a part property, the tool offers you a list of blocks, from which you can choose the part property type. If the necessary block doesn't exist yet, you can simply create it by typing its name directly on the part property shape.



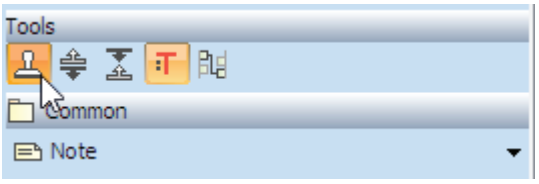
3. Click the Part Property button on the diagram palette and then click an empty place on the diagram pane. An unnamed part property is created, and the list of existing blocks to type it is offered.

4. Type Climate Control Unit next to the colon (":") directly on the part property shape and press Enter. The *Climate Control Unit* block to type the just-created part property is created in the Model Browser.

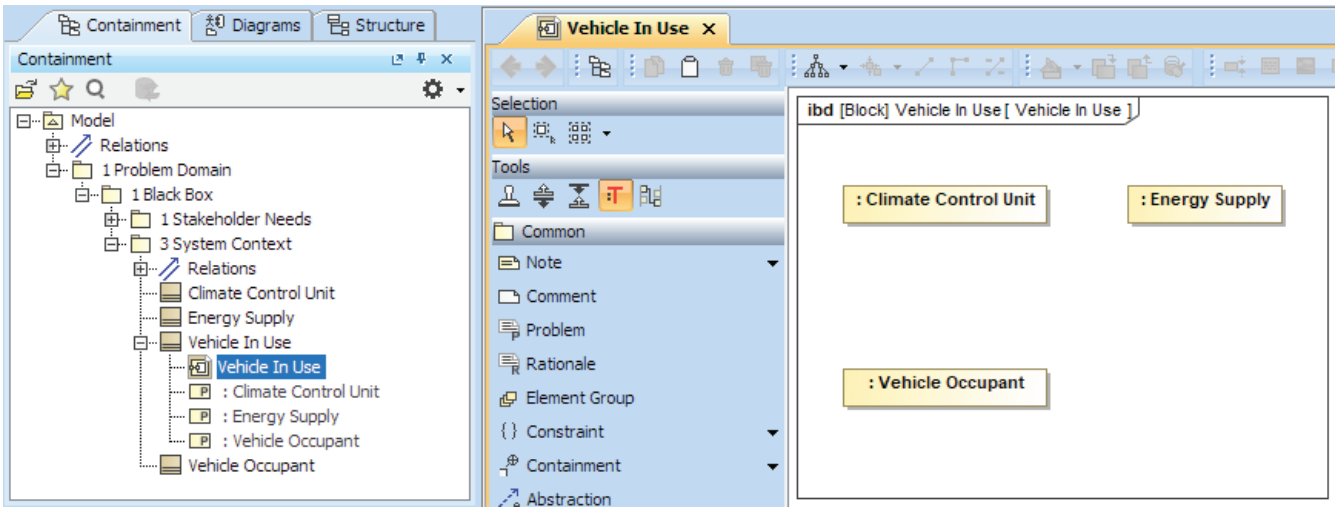
> ⓘ When the Type Selection Mode is enabled, typing on the part property shape always defines the name of the new block, which types that part property, but not the name of the part property. If you want to specify the name as well, type it before the colon (":") on the shape. However, na mes are not necessary, since part properties created in this diagram can be easily identified by their types.



5. Repeat steps 3 and 4 to create two more part properties: one typed by the *Vehicle Occupant* block and another typed by the *Energy Supply* block.

> ⓘ Since you need to create more than one part property in succession, use the functionality of the Sticky button (see the following figure) so that you don't need to click the **Part Property** button on the palette each time you want to create a new part property. When the Sticky mode is enabled, you can create as many elements of the selected type as you need. Press Esc when you're done.
>
> 
>

When you're done, you have three part properties and exactly the same number of blocks that type them in your model.



B2. Use Cases

What is it?

In this cell, functional stakeholder needs are refined with use cases and use case scenarios. In comparison to stakeholder needs, use cases are more precise in telling what people expect from the system and what they want to achieve by using it. All the use cases are grouped into system contexts defined in B3. System Context.initial.

Overall, this cell produces:

- Functional use cases that provide measurable value to the user.
- Use case scenarios on how the system interacts with the user and/or other systems.

Who is responsible?

In a typical multidisciplinary systems engineering team, use cases are specified by the *Requirements Team*.

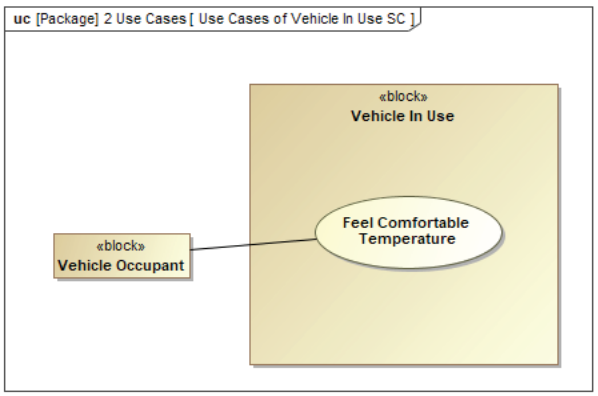
How to model?

In the modeling tool, use cases of the system can be captured by utilizing the infrastructure of the SysML use case diagram. Every use case diagram should be created considering one of the system contexts defined in B3. System Context.initial. Use cases can be captured as elements of the use case type. A single use case can be performed in different contexts.

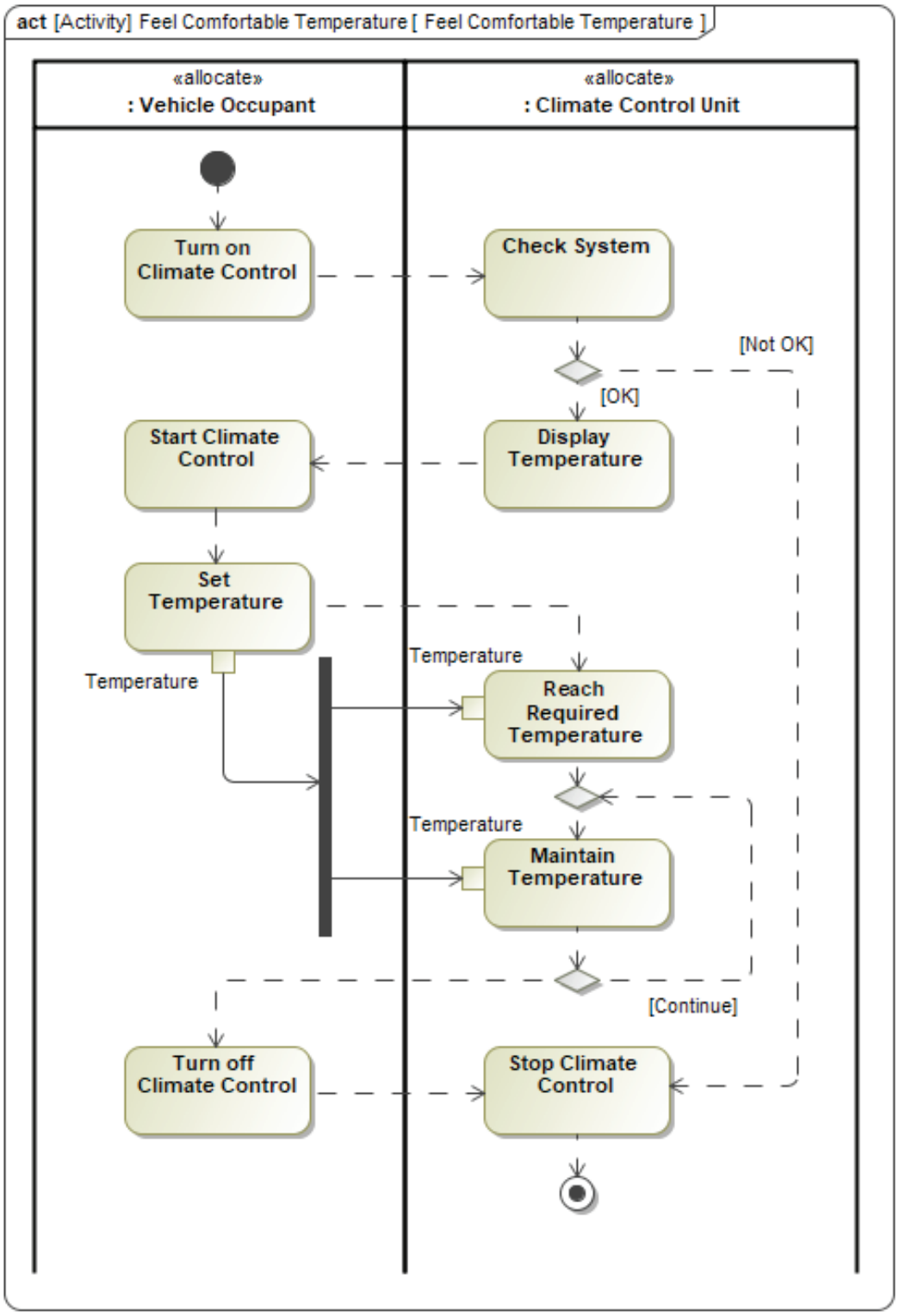
Entities that perform one or more use cases within the system context should be represented as blocks in the use case diagram. Created in the initial phase of B3, these blocks type part properties of the system context. As you can see in the following figure, even humans, such as the vehicle occupant, should be represented as blocks.

ⓘ We recommend using blocks instead of actors to define humans for the following reasons:

- A block is treated as a part of the system context. An actor is not.
- A block can be marked as external. An actor cannot.
- A block can have its behavior defined. An actor cannot.



Each use case must have a name and primary scenario. Alternative scenarios are optional. A use case scenario can be captured in a form of SysML activity or sequence diagram. In the activity diagram, SoI, supposed users of the system, and/or other systems can be represented as swimlane partitions. In the sequence diagram, they are represented as lifelines. The SoI is meanwhile considered a black box.



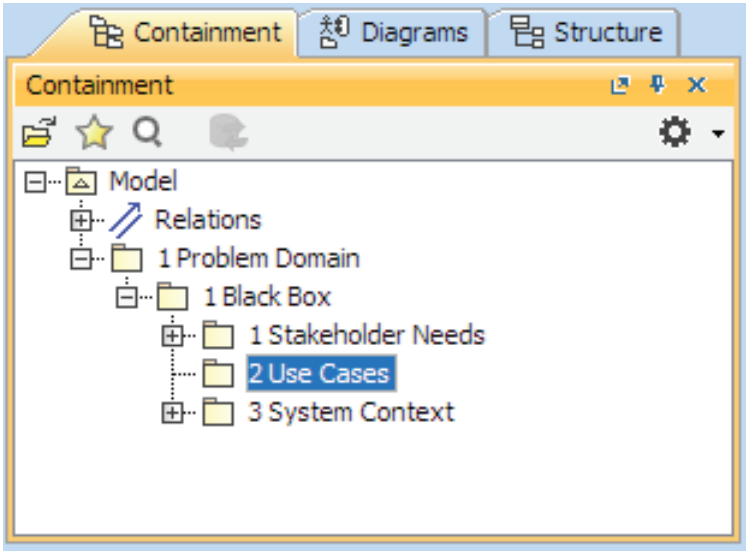
What's next?

- Now that you have use case scenarios, you can switch to B3.final. Use case scenarios serve as input for specifying interactions between the SoI and its environment in a variety of system contexts.
- Primary use case scenarios, describing interactions between the system and some external entities within its environment, are decomposed to specify the internal action/sequence flows among the subsystems in W2.initial.

Tutorial

Step 1. Organizing the model for B2

Following the structure of the MagicGrid framework, use cases and their scenarios in the form of SysML activity or sequence diagrams, together with elements they represent, should be stored in a separate package inside the *1 Black Box* package. We recommend naming the package after the cell, that is, *2 Use Cases*.



To organize the model for B2

1. Right-click the *1 Black Box* package and select **Create Element**.
2. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.
3. Type *2 Use Cases* to specify the name of the new package and press Enter.

Step 2. Creating a diagram for use cases

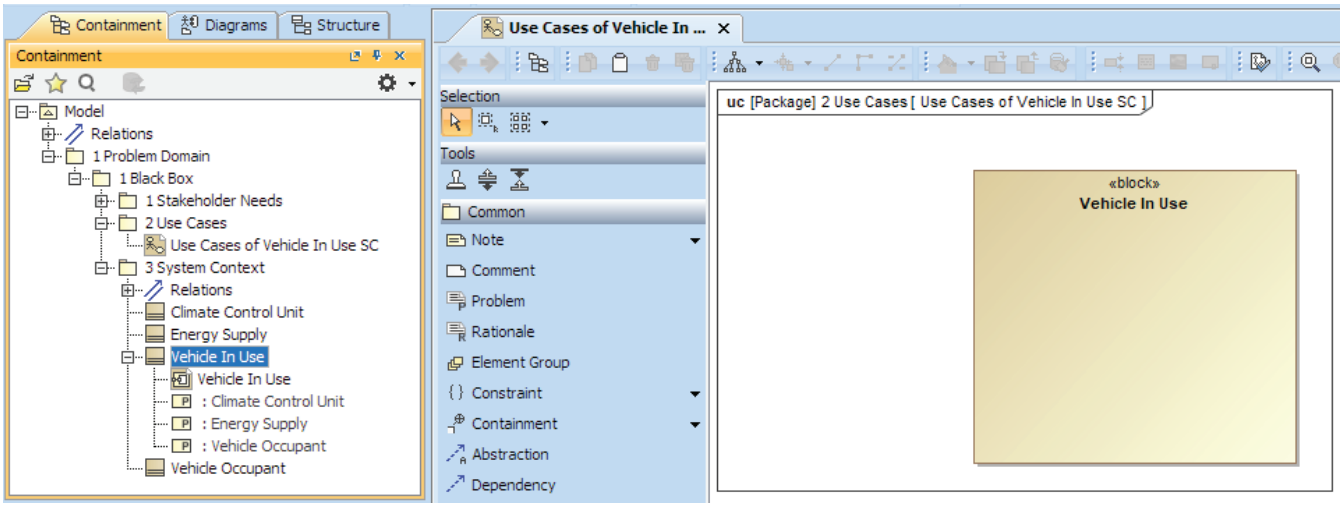
To start capturing use cases of the *Vehicle In Use* system context (defined in step 2 of the B3 initial phase tutorial), you need to create a SysML use case diagram first. You also need to display the shape of the block that captures the system context on the diagram pane. The *Vehicle Occupant block*, which represents the user of the SoI, should also be displayed on the diagram pane.

To create a diagram for capturing use cases performed in the *Vehicle In Use* system context

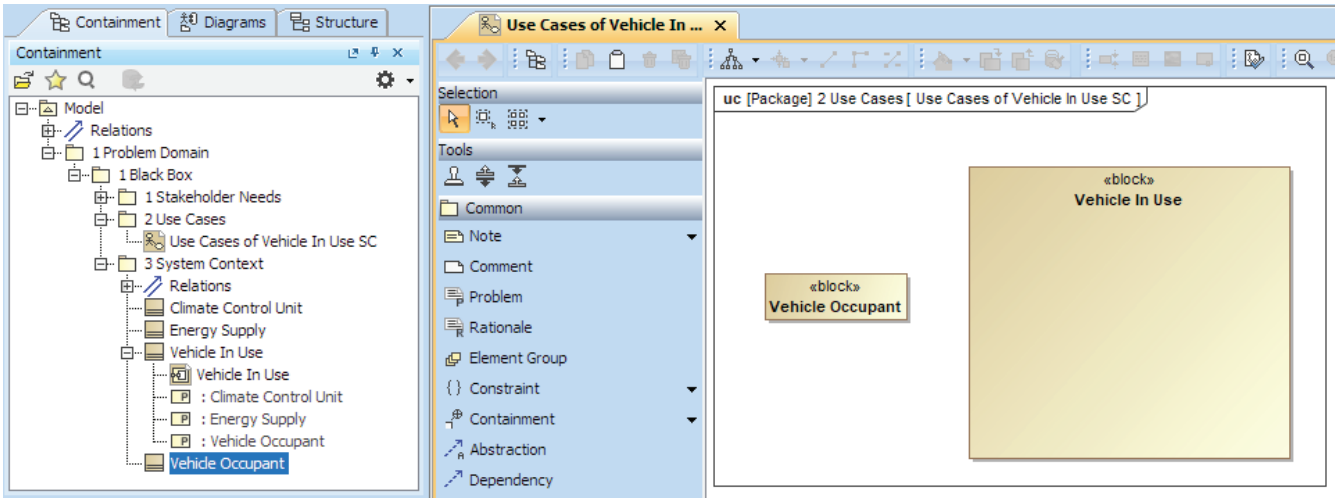
1. Right-click the *2 Use Cases* package and select **Create Diagram**.
2. In the search box, type *uc*, where *u* stands for *use* and *c* for *case*, and then press Enter. The diagram is created.

   ⓘ If you have more than one system context, we recommend storing use cases of each system context in a separate package. In this case, there is no such need.
3. Type *Use Cases of Vehicle In Use SC* to specify the name of the new diagram and press Enter again.

4. In the Model Browser, select the *Vehicle In Use* block and drag it to the newly created use case diagram pane. The shape of the *Vehicle In Use* block appears on the diagram.



5. In the Model Browser, select the *Vehicle Occupant* block and drag it to that diagram pane as well. The shape of the *Vehicle Occupant* block appears on the diagram too.



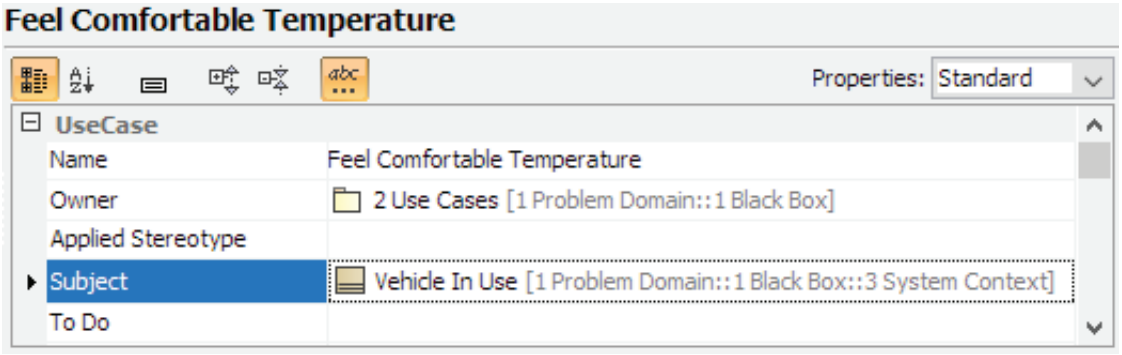
Step 3. Capturing use cases

By analyzing stakeholder needs, we can identify the *Feel Comfortable Temperature* use case performed in the *Vehicle In Use* system context. Now let's capture it in the *Use Cases* of *Vehicle In Use SC* diagram.

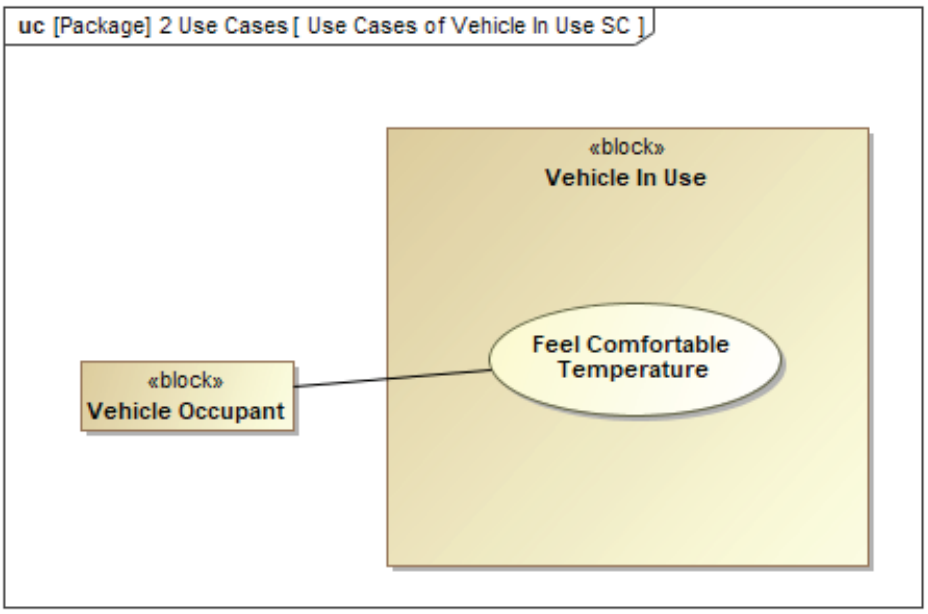
To capture the *Feel Comfortable Temperature* use case

1. Open the use case diagram created in step 2 of the B2 tutorial, if not yet opened.
2. Click the **Use Case** button on the diagram palette and move the mouse over the shape of the *Vehicle In Use* block.
3. When you see the blue border around the shape of the *Vehicle In Use* block, click it. An unnamed use case is created within the shape of the block.
4. Type *Feel Comfortable Temperature* to specify the name of this use case and press Enter.

5. Double-click the use case to open its Specification and make sure that the *Vehicle In Use* block is set as its subject.



6. Select the *Feel Comfortable Temperature* use case, click the Association button on its smart manipulator toolbar, and then select the *Vehicle Occupant* block, which represents the user of the system, to relate these two elements.



Step 4. Creating a diagram for specifying the use case scenario

A use case scenario can be captured in a SysML activity or sequence diagram. We choose the former one.

To create a SysML activity diagram for specifying the *Feel Comfortable Temperature* use case scenario

1. In the *Use Cases of Vehicle In Use SC* diagram, right-click the shape of the *Feel Comfortable Temperature* use case and select **Create Diagram**.

2. In the search box, type *act*, the acronym of the SysML activity diagram, and press Enter. The diagram is created and opened. It is owned by the SysML activity with the same name.

> ⓘ Now if you get back to the use case diagram, which displays the *Feel Comfortable Temperature* use case, you can see that the shape of that use case is decorated with the rake ( ) icon. The decoration means that the use case contains an internal diagram, which is the *Feel Comfortable Temperature* activity diagram. Double-clicking the shape of the use case opens that internal diagram.
>
> 
>

Step 5. Specifying the use case scenario

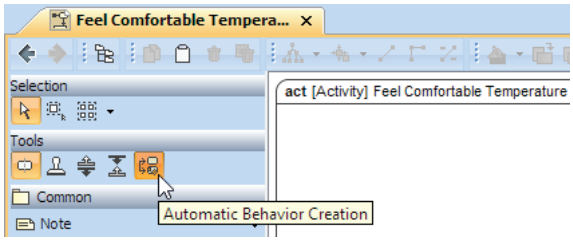
Let's say the *Feel Comfortable Temperature* use case scenario includes these steps:

1. Turn on Climate Control
2. Check System
3. Display Temperature
4. Start Climate Control
5. Set Temperature
6. Reach Required Temperature
7. Maintain Temperature
8. Turn off Climate Control
9. Stop Climate Control

To specify the basic use case scenario

1. Open the *Feel Comfortable Temperature* activity diagram (created in step 4 of the B2 tutorial), if not yet opened.
2. Be sure the Automatic Behavior Creation mode is enabled in the diagram (see the following figure). Otherwise, actions created in the diagram will not be typed by activities.

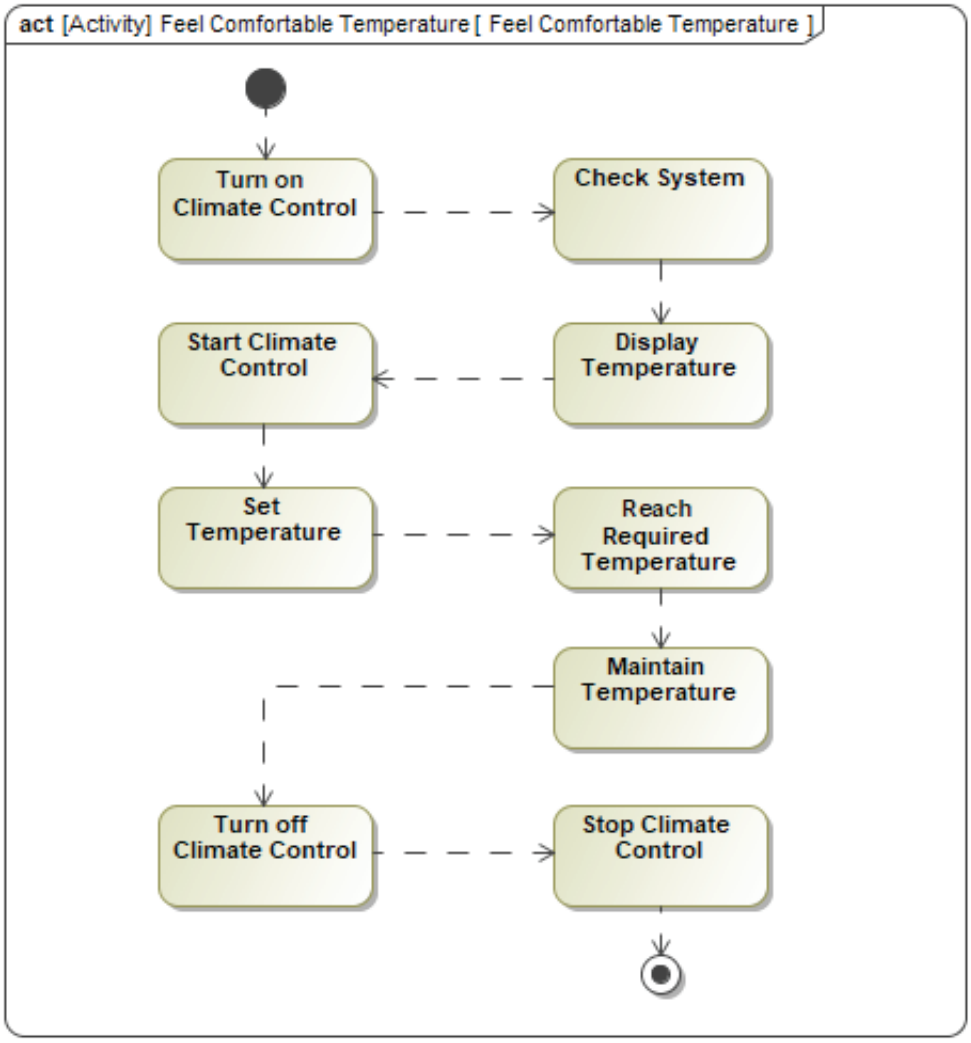
> ⓘ Activities are necessary for functional decomposition performed in W2.



3. Click the **Initial Node** button on the diagram palette and then click an empty place on the diagram pane. The initial node is created and displayed on the diagram.
4. Click the Control Flow button on the smart manipulator toolbar of the initial node and then click an empty place on the diagram pane. A unnamed action typed a new activity is created.

5. Click the shape of the action and type *Turn on Climate Control* after the colon (“**:**”) to name the activity. Press Enter after you’re done.
6. Repeat steps 4 and 5 to capture other steps from the list above.
7. Select the shape of the last action and click the Control Flow button on its smart manipulator toolbar.
8. Right-click an empty place on the diagram pane and select **Activity Final**. The final node is created and displayed on the diagram.

When you’re done, your *Feel Comfortable Temperature* diagram should look very similar to the one in the following figure.



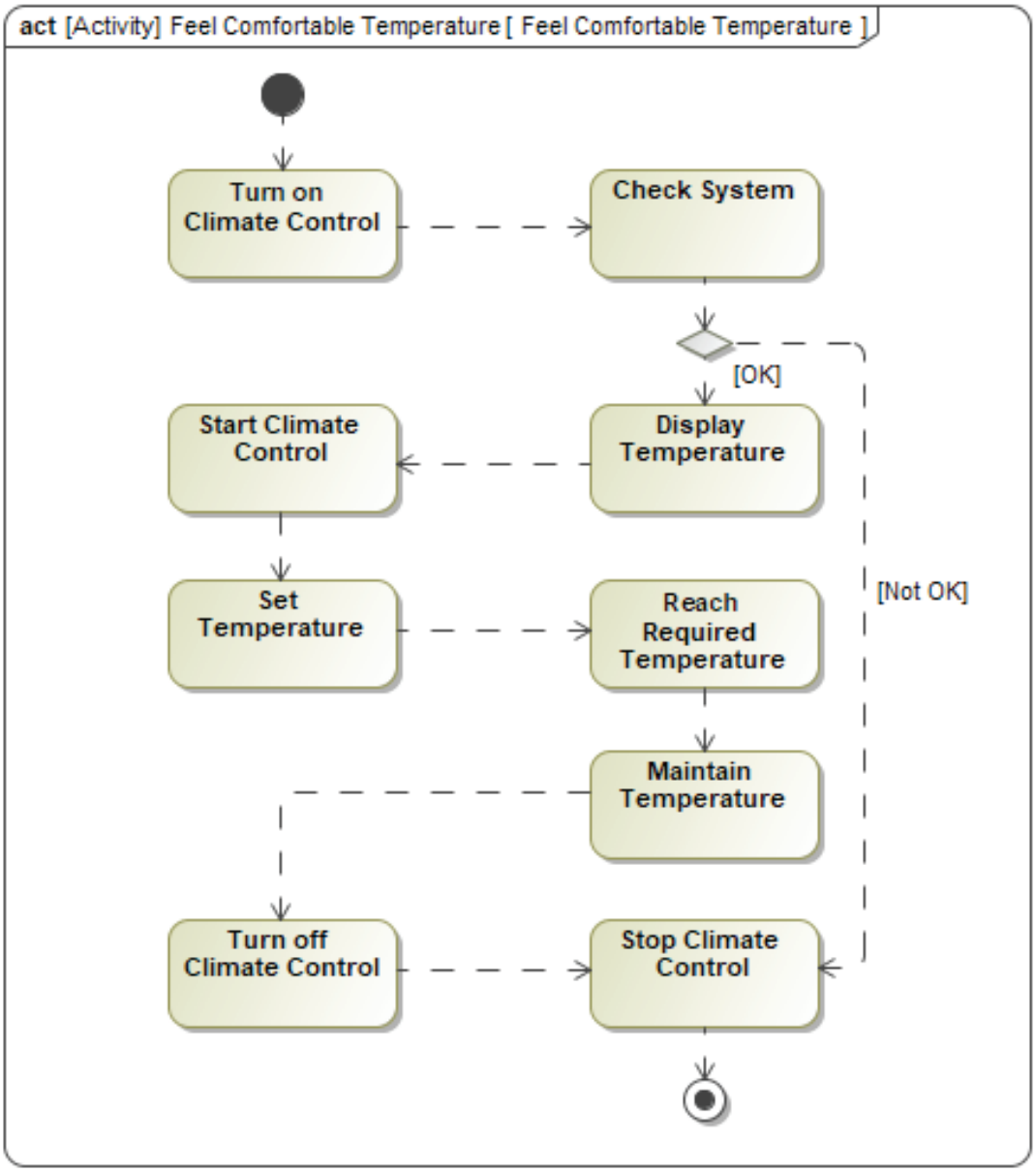
Now let’s say you want to make the scenario more complex by adding an alternative flow of actions to specify what happens if the Climate Control Unit doesn’t work. For this, you need to insert a decision node between the *Check System* and *Display Temperature* actions, as well as draw two more control flows with appropriate guards.

To update the use case scenario with an alternative flow

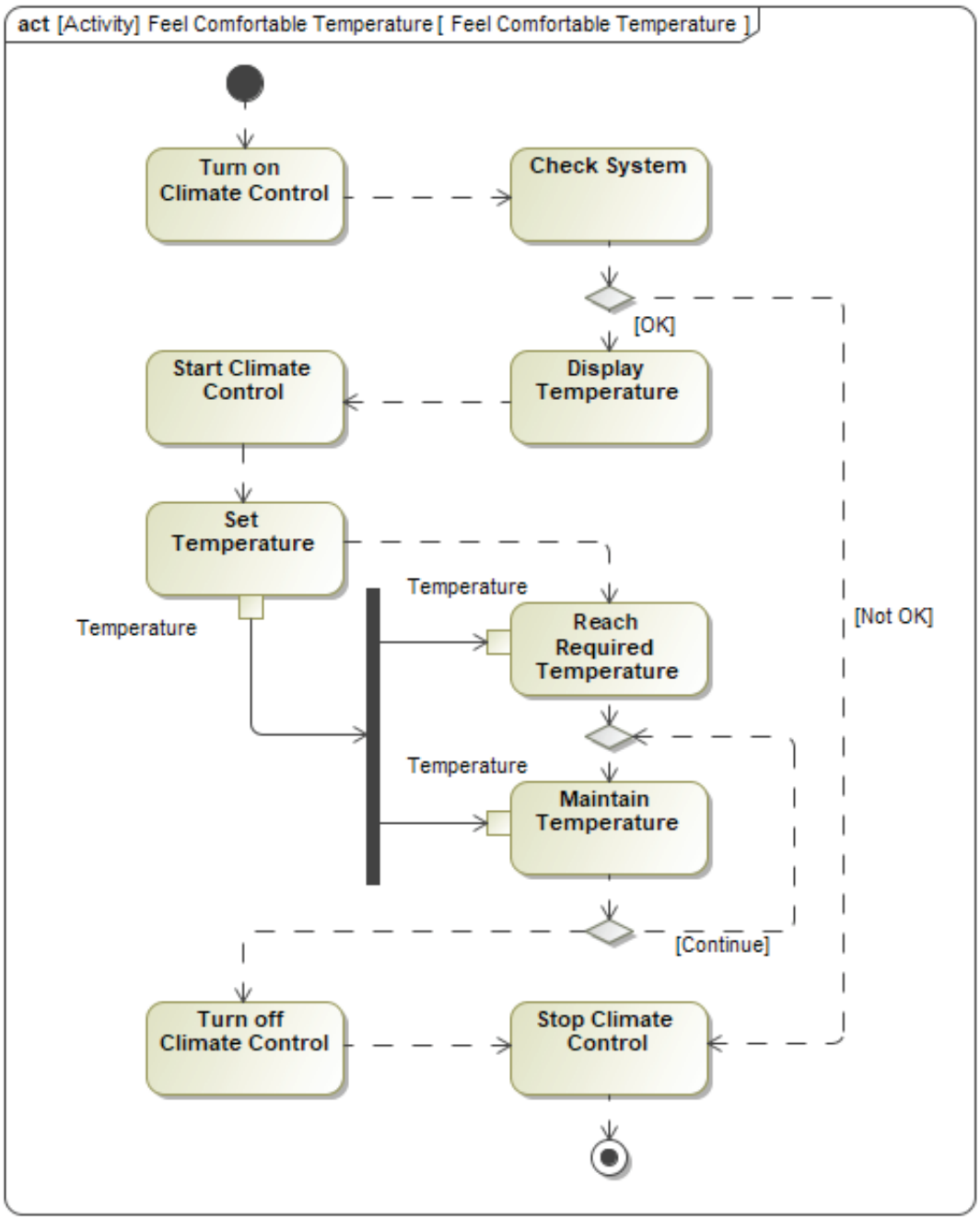
1. Click the **Decision** button on the diagram palette and then click the control flow between the *Check System* and *Display Temperature* actions.

2. Click either **After Control Flow** or **Before Control Flow**. The decision node is inserted.
3. Select the control flow between the decision node and the *Display Temperature* action, type *[OK*, and press Enter. The *[OK]* guard is specified for this control flow.
4. Select the decision node, click the Control Flow button on its smart manipulator toolbar, then click *Stop Climate Control* action. The control flow is created.
5. Select that control flow, type *[Not OK*, and press Enter. The *[Not OK]* guard is specified for this control flow.

After you're done, your *Feel Comfortable Temperature* diagram should look very similar to the one in the following figure.



A few more items should be added to this diagram to make the use case scenario complete (see the following figure). It's not discussed here how to do this, but you can analyze the latest documentation of the modeling tool on how to insert the fork node and pins.



Step 6. Allocating actions to participants of the system context

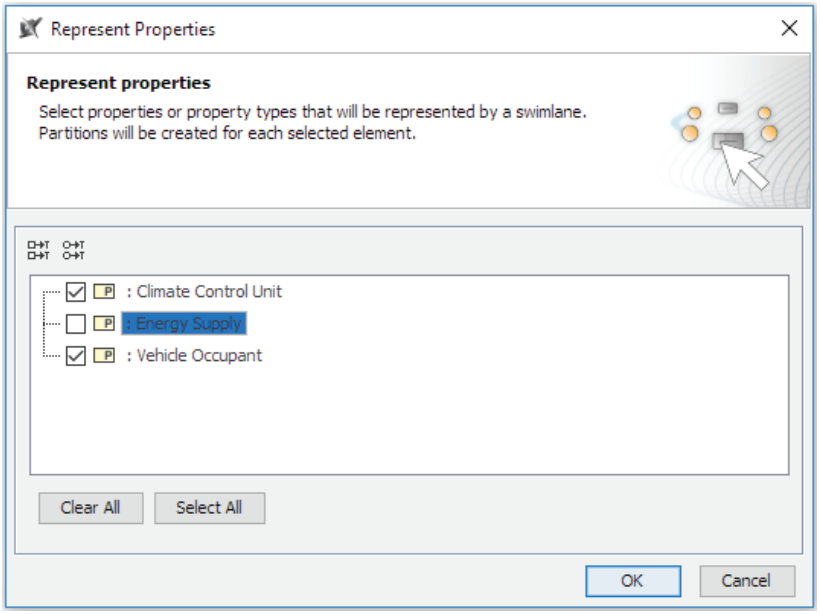
After you have the entire scenario of the *Feel Comfortable Temperature* use case in your model, you can specify which participant of the related system context is responsible for performing which action. There is a generic term for this: allocating behavior to structure.

You can use swimlane partitions to represent participants of the system context. In this case, you need two of them, and they are captured in the model as part properties of the *Vehicle In Use* system context: one part property is typed by the *Vehicle Occupant* block, and the other by the *Climate Control Unit* block (see step 4 of the B3 initial phase tutorial).

Before establishing allocations, you need to make sure that the allocation to usage mode is enabled in your activity diagram. This mode allows you to convey that allocations are established considering the system context. Otherwise, allocations are generic, unrelated to any system context.

To allocate behavior to structure in the *Vehicle In Use* system context

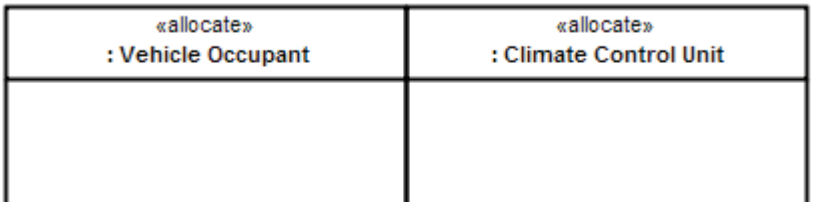
1. Click the **Vertical Swimlanes** button on the use case diagram palette and then click an empty space on that diagram pane. The **Represent Properties** dialog opens.
2. Since all the items are by default selected in the dialog, click to clear the part property typed by the *Energy Supply* block (see the following figure). You don't need to allocate any action in this diagram to this part property.



> ⓘ The dialog always displays the list of part properties that belong to the subject of the use case, which encloses the activity diagram. In this case, the part properties of the *Vehicle In Use* block (see step 4 of the B3 initial phase tutorial) are displayed.

3. Click the **OK** button on the dialog. The swimlanes are displayed on the diagram.

> ⓘ Swimlane partitions are always arranged alphabetically from A to Z. Thus, the Climate Control Unit partition appears by default on the left side, and the *Vehicle Occupant* on the right. If you want the *Vehicle Occupant* partition to be on the left (see the following figure), select the header of the Climate Control Unit partition and click the Move Swimlane Right button ▶. As a result, swimlane partitions become swapped.

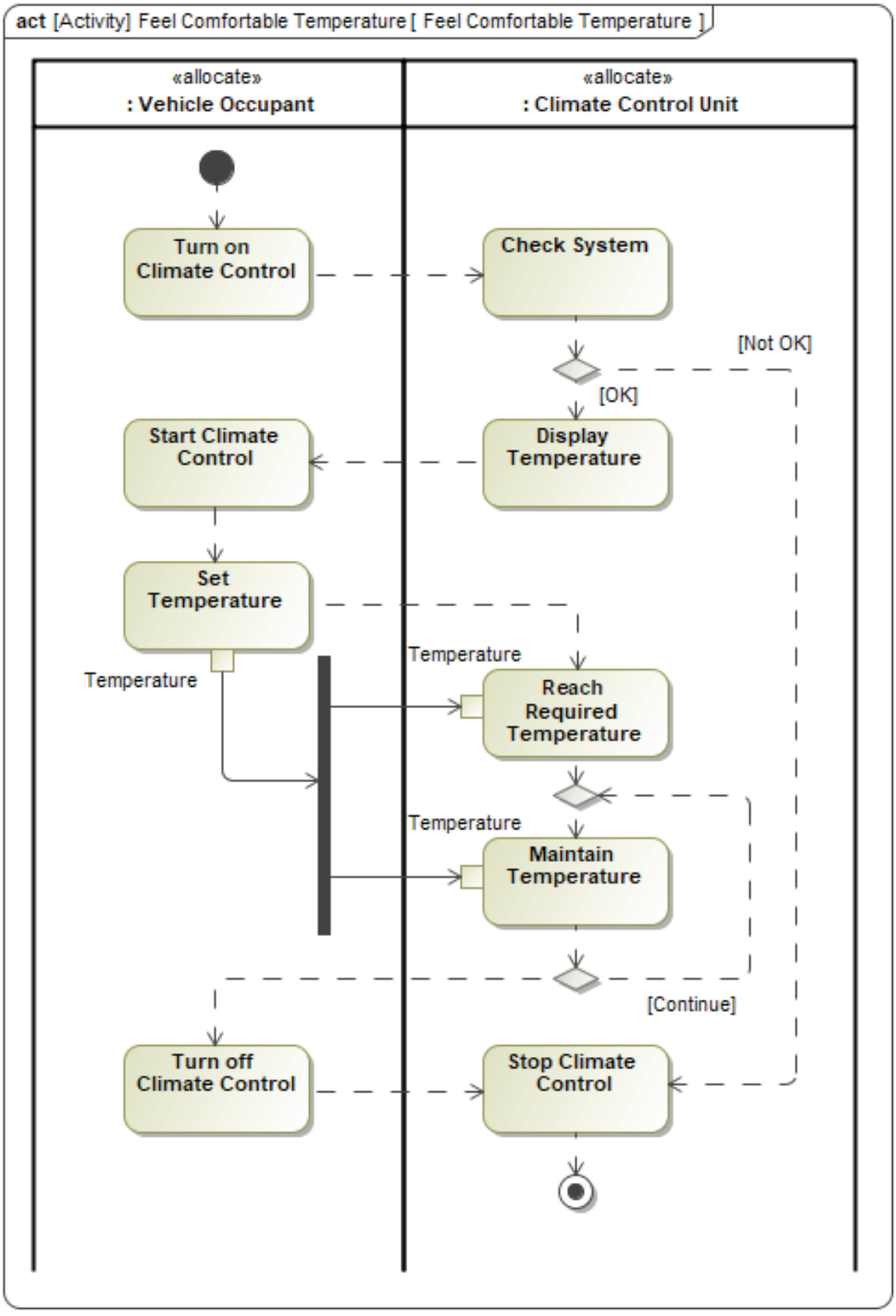


4. Right-click any of the swimlane partitions and select **Allocation Mode** > **Usage** (if not selected yet). The allocation to usage mode is enabled in the diagram.

> ⓘ This mode allows you to convey that allocations are established considering the system context. Otherwise, allocations are generic, not related to any system context.

5. Drag actions of the vehicle occupant to the *Vehicle Occupant* swimlane.
6. Drag actions of the climate control unit to the Climate Control Unit swimlane.

When you're done, your activity diagram *Feel Comfortable Temperature* should look very similar to the one below.



B3. System Context: final

What is it?

System context or operating environment determines an external view of the system. It must introduce elements that do not belong to the system but do interact with that system. More than one system context can be defined for a single SoI. Besides the system itself (currently considered to be a black box), the collection of elements in the particular system context can include external systems and users (humans, organizations) that interact with the SoI in that context.

Modeling in this cell includes two phases: initial and final. During the initial phase of B3, one or more system contexts are defined, and elements that participate in each context are identified. In the final phase of B3, interactions within each system context are specified. Since this requires analyzing system behavior, you should identify use cases of that system context and model use case scenarios first. That's why the B3 cell is interrupted by the B2 cell.

The final phase of the B3 cell produces:

- Relations between participants of every system context defined in the initial phase of B3
- Items that flow via these relations

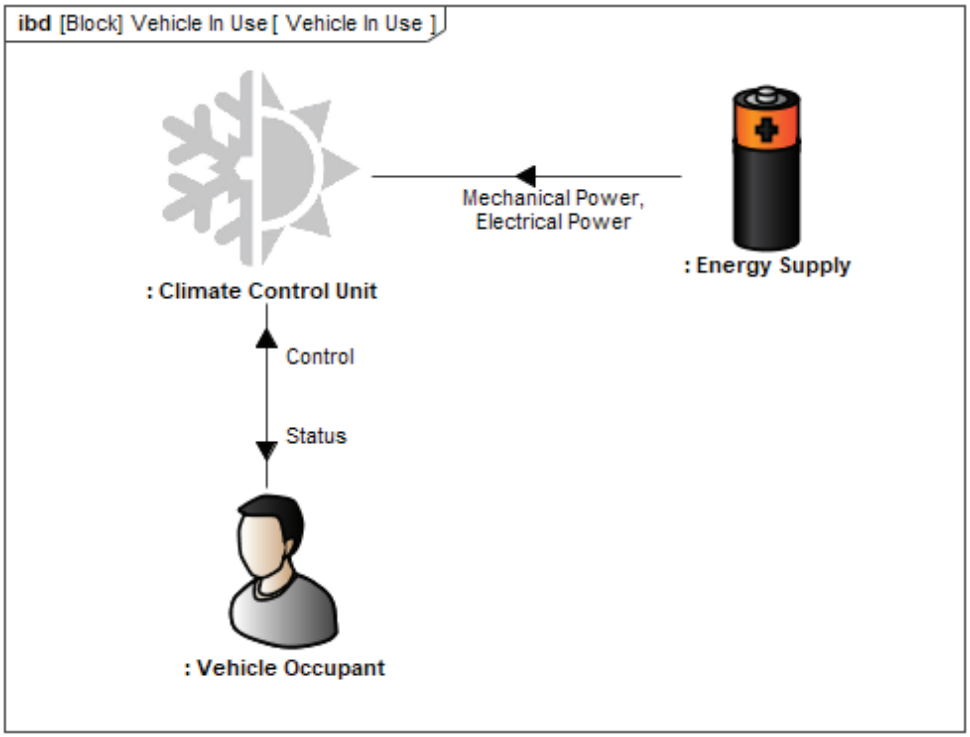
Who is responsible?

In a typical multidisciplinary systems engineering team, the interactions within each system context are specified by the *Requirements Team*.

How to model?

In the modeling tool, relations between the participants of the system context can be captured as connectors between the part properties that represent these participants. Once the connector is created, it is necessary to specify one or more items flowing from one end of the connector to another, because the connector alone says nothing about the nature of the interaction. It can be a physical, informational, or control flow. Items can be stored in the model as signals, blocks, or value types, depending on their nature.

As you can see in the following figure, system context diagrams can be supplemented with various images related to a SoI so that the result could be presented to stakeholders, including even those who are incapable of reading models.



What's next?

- After the block that captures the system of interest is created, quantitative characteristics of the system describing non-functional stakeholder needs can be captured. Thus, you can switch to B4. Measurements of Effectiveness.

Tutorial

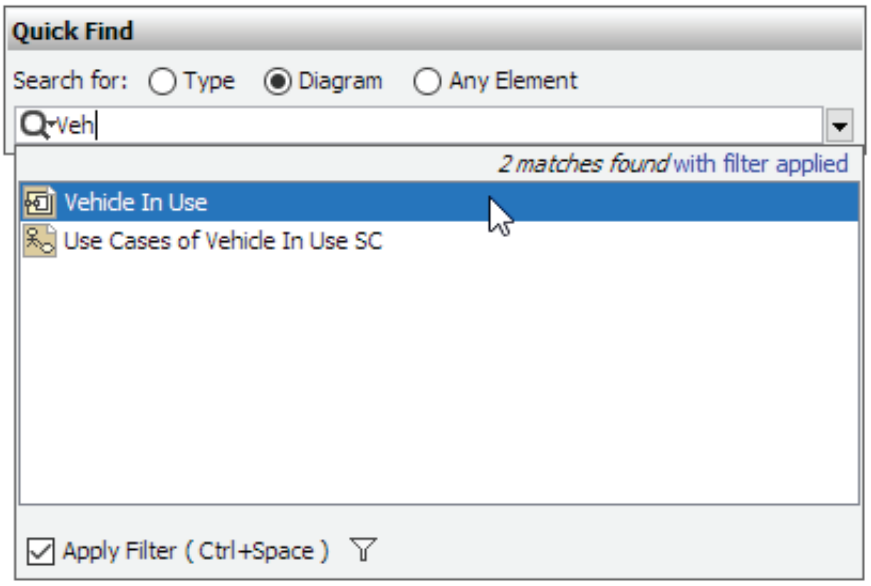
Step 1. Specifying interactions between the participants of system context

Now it's time to update the ibd of the *Vehicle In Use* system context (see step 4 of the B3 initial phase tutorial). For this, you should analyze the scenario of the related use case, that is, *Feel Comfortable Temperature*. To be more precise, you should consider the control and object flows that cross the boundary between swimlane partitions, which represent the part properties of that system context. The use case scenario reveals how these part properties are related to one another: the vehicle occupant controls the Climate Control Unit, and the unit provides him/her with information, such as status and cabin temperature.

Let's start by drawing a connector to represent the interaction.

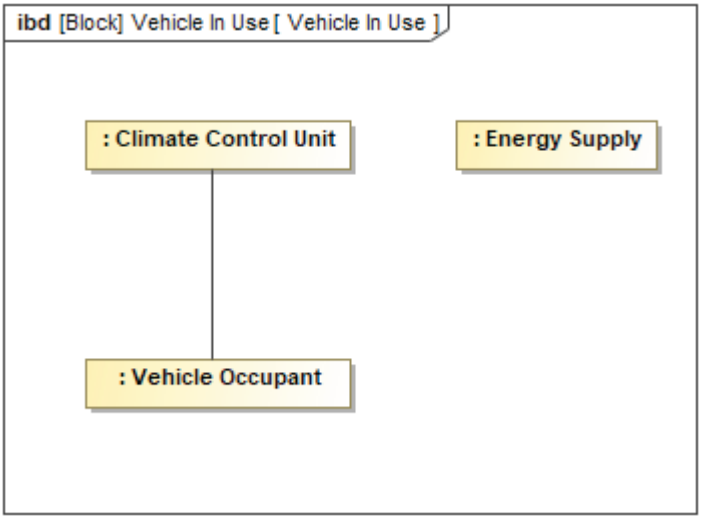
To draw the connector between the part property typed by the *Vehicle Occupant* block and the one typed by the *Climate Control Unit* block

1. Open the *Vehicle In Use* ibd:
   a. Press Ctrl + Alt + F to open the **Quick Find** dialog.
   b. Click **Diagram** to search for diagrams only and type *Veh*.
   c. When you see the *Vehicle In Use* ibd selected in the search results list (see the following figure), press Enter. The selected diagram opens.

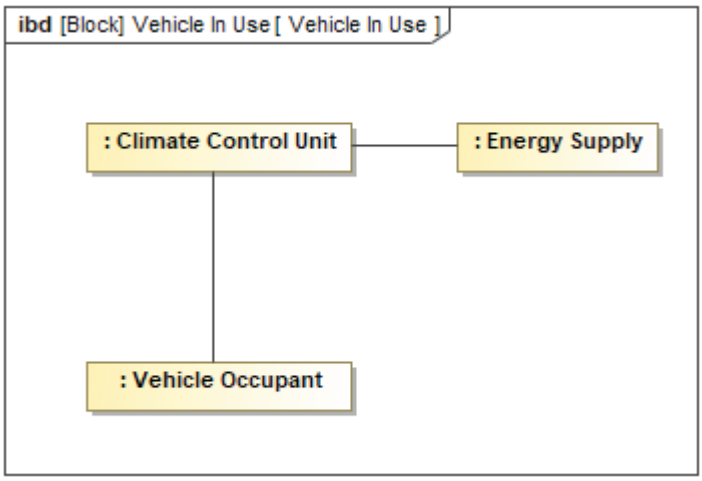


2. Select the part property typed by the *Vehicle Occupant* block and click the Connector button on its smart manipulator toolbar.

3. Select the part property typed by the *Climate Control Unit* block. The connection is created.

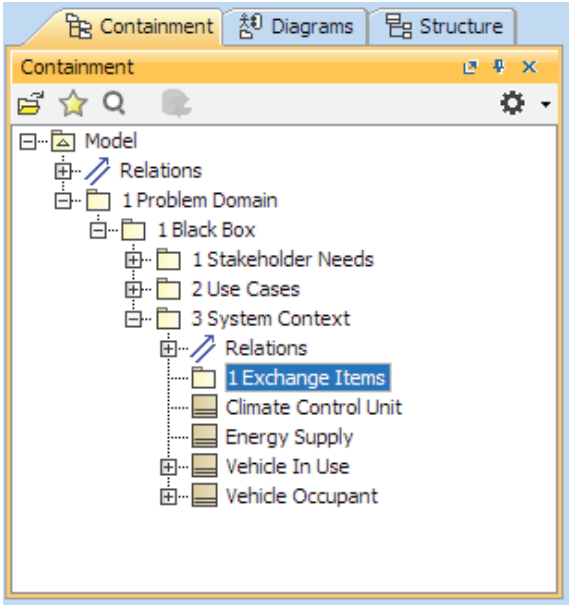


You can also create a connector between the part property typed by the *Energy Supply* block and the one typed by the *Climate Control Unit* block. Once you're done, your ibd should look very similar to the one below.



Step 2. Assigning items flowing between the participants of system context

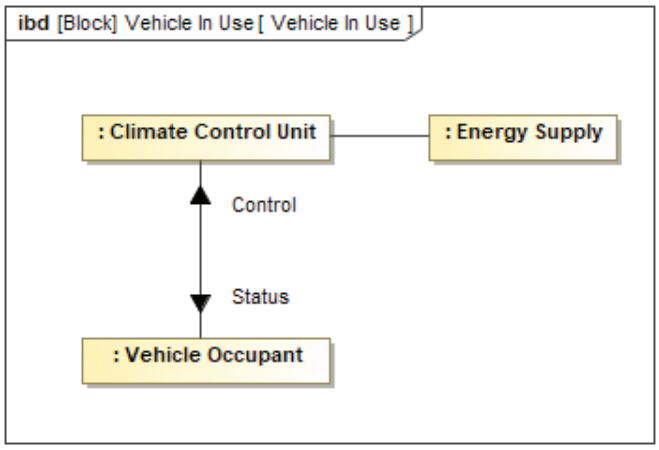
Now, let's create items flowing from one end of the connector to another and backwards. We recommend storing elements that capture these items in a separate package within the *3 System Context* package.



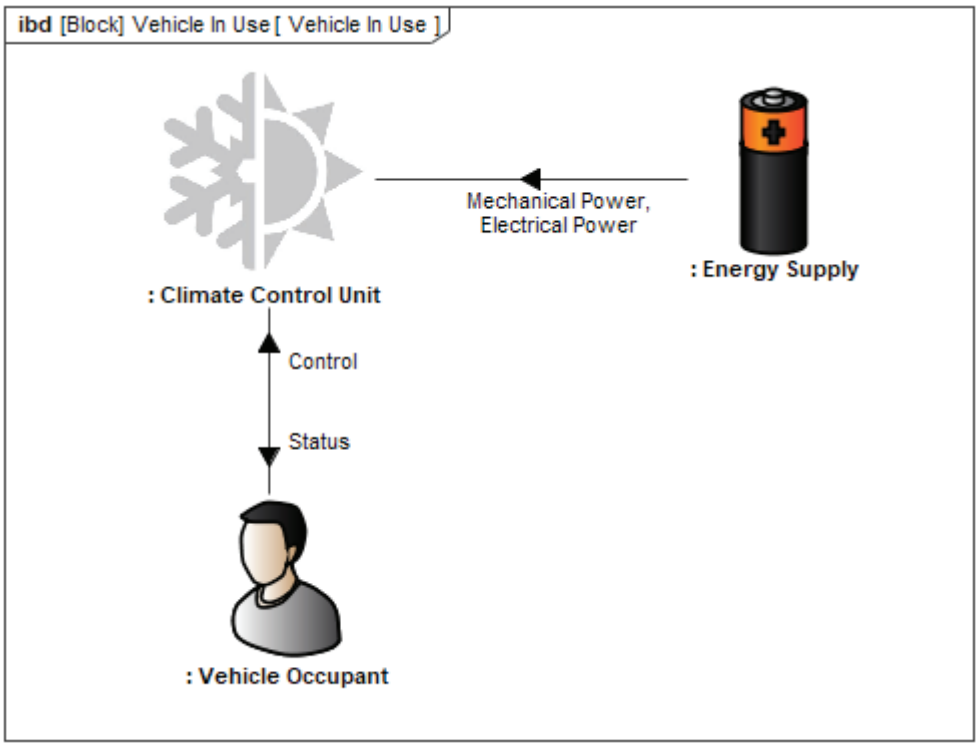
If we assume that you have created the *1 Exchange Items* package on your own, we can move directly to creating items that flow and assigning them to connectors. One or more item flows can be assigned on a single connector.

To capture item flows and assign them to the connector between part properties

1. Right-click the *1 Exchange Items* package and select **Create Element**.
2. In the search box, type *si*, the first two letters of the element type *Signal*, and press Enter.
3. Type *Control* to specify the name of the new signal and press Enter.
4. Drag the signal to the connector between the part property typed by the *Climate Control Unit* block and the one typed by the *Vehicle Occupant* block in the *Vehicle In Use* ibd.
5. In the open dialog, select **From :Vehicle Occupant To :Climate Control Unit** as direction of the flow and click **OK**. The *Control* signal is assigned to the connector.
6. Repeat steps 1 to 3 to create the signal named *Status*.
7. Drag the signal to the same connector in the *Vehicle On* ibd.
8. In the open dialog, select **From :Climate Control Unit To :Vehicle Occupant** as direction of the flow and click **OK**. The *Status* signal is assigned to the connector.



You can also assign an item flow between the part property typed by the *Energy Supply* block and the one typed by the *Climate Control Unit* block. Items that flow can be electrical and mechanical power captured in the model as blocks. Once you're done, your ibd should look very similar to the one below.



ⓘ As you can see in the figure above, system context diagrams can be supplemented with various images related to a SoI so that the result could be presented to stakeholders, including even those who are incapable of reading models. For more information, see the latest documentation of the modeling tool.

B4. Measurements of Effectiveness

What is it?

In this cell, non-functional stakeholder needs are refined with measurements of effectiveness (MoEs), which capture characteristics of the SoI in numerical format. MoEs is a traditional term widely used in systems engineering, describing how well a system carries out a task within a specific context. In the problem domain model, they serve as the high level key performance indicators that would be automatically checked within the solution domain model.

The same set of MoEs can be used in different models to specify numerical characteristics of other systems of interest.

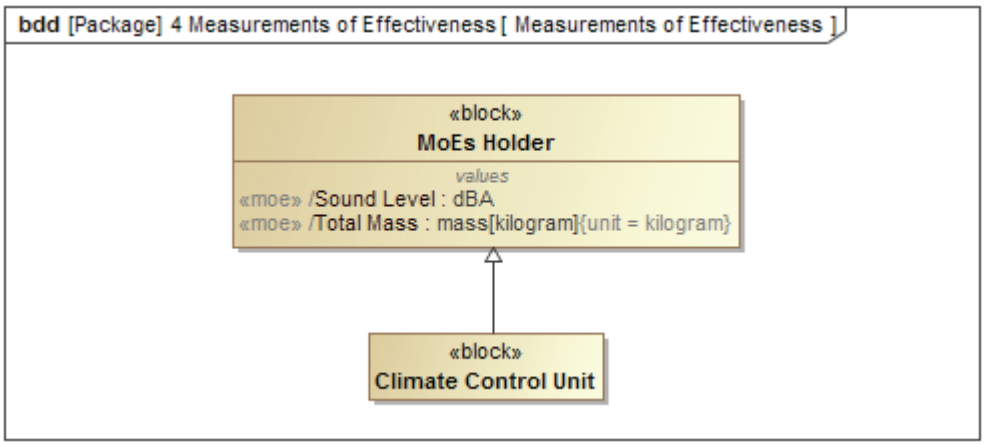
Who is responsible?

In a typical multidisciplinary systems engineering team, MoEs are specified by the *Requirements Team*.

How to model?

To capture MoEs in the modeling tool, a SysML block definition diagram (bdd) can be used. To define a reusable set of MoEs, a separate block should be created and set as a super-type of the block that stands for the SoI. In real-world projects, blocks with reusable sets of MoEs are stored in external models, also known as libraries. MoEs are captured as value properties with «moe» stereotypes applied. The block of the SoI inherits them from the super-type block. A mechanism of

redefinition in the modeling tool allows you to define different default values and refine different requirements by every single MoE.



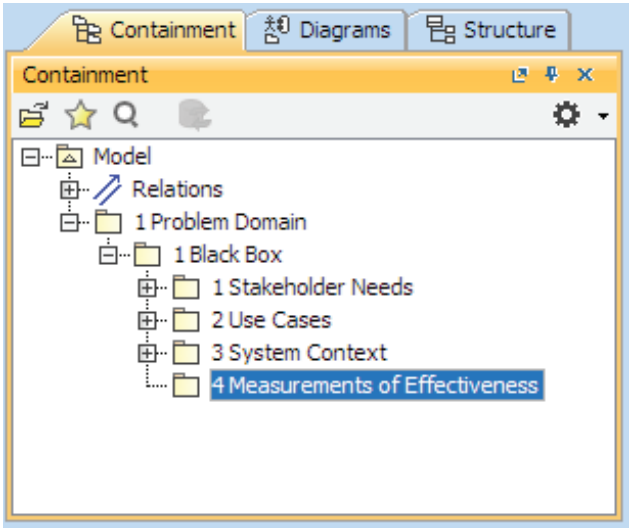
What's next?

- Now that you have measurements of effectiveness, you can specify design constraints, which can be used to verify whether the system design is right or not (see S4).

Tutorial

Step 1. Organizing the model for B4

Following the structure of the MagicGrid framework, model elements that capture the measurements of effectiveness should be stored in a separate package inside the *1 Black Box* package. We recommend naming the package after the cell, that is, *4 Measurements of Effectiveness.*



To organize the model for B4

1. Right-click the *1 Black Box* package and select **Create Element**.
2. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.
3. Type *4 Measurements of Effectiveness* to specify the name of the new package and press Enter.

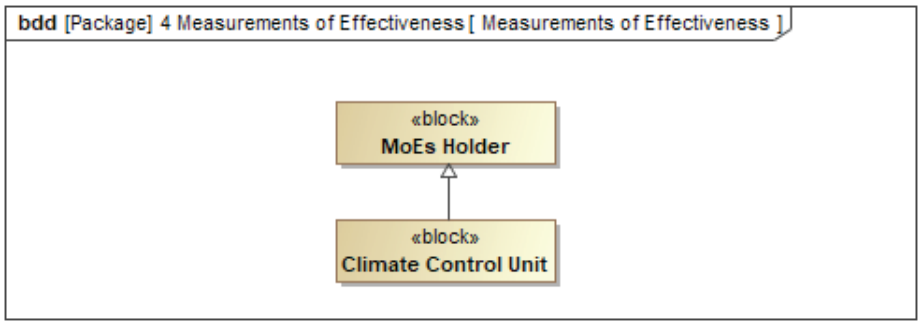
Step 2. Creating a block for capturing MoEs

To start capturing MoEs of the Vehicle Climate Control Unit, you first need to create a bdd and a block to set as a super-type of the *Climate Control Unit* block. Remember that if it was a real-world project, the MoEs holder would be created in some external model, also known as a library.

To create a diagram and a block for capturing MoEs

1. Right-click the *4 Measurements of Effectiveness* package and select **Create Diagram**.
2. In the search box, type *bdd*, the acronym of the SysML block definition diagram, and then double-press Enter. The diagram is created.

   ⓘ Note that the diagram is named after the package where is it stored. This name perfectly suits the diagram, too. You need only remove the sequence number from its name.
3. Click the **Block** button on the diagram palette and then click the empty space in that diagram pane. An unnamed block is created.
4. Type *MoEs Holder* to specify the name of this block and press Enter.
5. In the Model Browser, select the *Climate Control Unit* block and drag it to the diagram pane. The shape of the *Climate Control Unit* block appears on the diagram.
6. Click the Generalization button on the smart manipulator toolbar of the *MoEs Holder* block and then click the shape of the *Climate Control Unit* block. The *MoEs Holder* block becomes the super-type of the *Climate Control Unit* block.



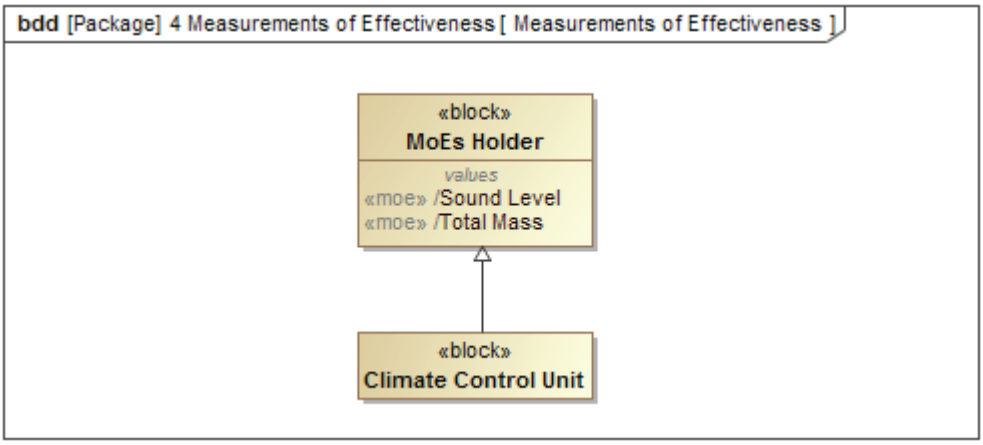
Step 3. Capturing MoEs

Now let's analyze non-functional stakeholder needs *SN-1.3 Noise Level* and *SN-1.4 Climate Control Mass. SN-1.3* prompts you to create the *Sound Level* MoE, and the *SN-1.4* determines the *Total Mass* MoE. Let's capture these MoEs in your model.

To capture MoEs of the Vehicle Climate Control Unit

1. Select the shape of the *MoEs Holder* block and click the **Create Element** button > **Value Property**.
2. Type *Sound Level* to specify the name of the new value property and press Enter.
3. Right-click the value property and select **Stereotype**.
4. In the search box, type *moe* and press Ctrl + Spacebar to select the «moe» stereotype.
5. Press Enter. The «moe» stereotype is applied on the *Sound Level* value property.
6. Right-click the value property and select **Is Derived**. It becomes derived, which means that it is calculable (not definable).

7. Repeat the entire procedure to capture the *Total Mass* MoE.



In addition to this, you should specify value types for these MoEs. The majority of the commonly used value types are stored in the ISO 80000 library of units.

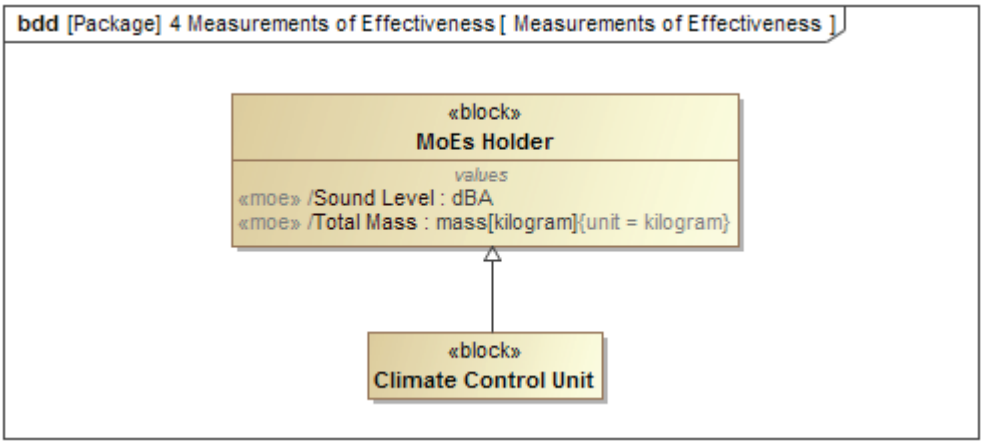
To set a standard unit (from the library) as MoE type

1. Select any of the value properties and click **ISO** on the smart manipulator toolbar. Wait until the library of units is loaded.
2. Select the *Total Mass* value property and click the Specify Type button on the smart manipulator toolbar.
3. Type *kilog* to find the *mass[kilogram]* value type and select it. The type is specified.

Custom units should be created manually.

To set a custom unit as MoE type

1. Select the *Sound Level* value property and click the Specify Type button on the smart manipulator toolbar.
2. Type *dBA* and press Enter. The custom value type is created in the model, directly under the package *4 Measurements of Effectiveness,* and specified as the value type of the *Sound Level* value property.



ⓘ The set of MoEs captured as the *MoEs Holder* block can later be used in different models to assign the same numerical characteristics to other systems of interest.

White-box perspective

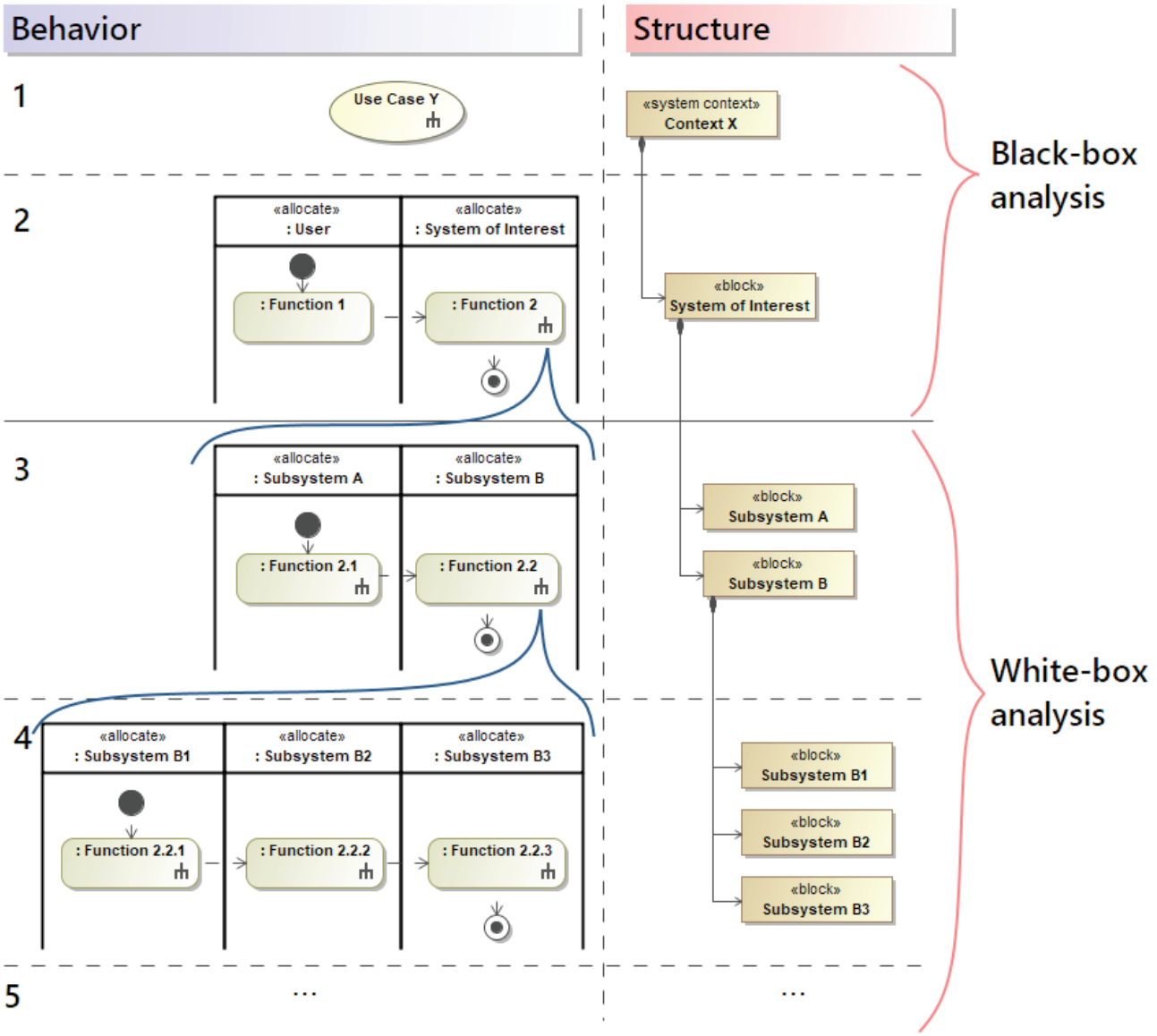
Introduction

Once the problem definition from the black-box perspective is finished, or in other words the operational analysis of the SoI is completed, you can switch to analyzing the SoI from the white-box perspective. This helps to understand in detail how the system shall operate, instead of already producing the system design.

In the second phase of the problem domain definition, you should go into deeper analysis of system functions in order to identify the functional blocks, also known as logical subsystems of the SoI. Functional blocks are the first step towards the solution architecture of the system.

Deeper analysis of system functions is called functional decomposition. As you can see in the following figure, every function performed by the SoI, as identified in the B2 cell (layer 2), is decomposed in the W2.initial cell (layer 3). Functions of the detailed behavior are then grouped by functional blocks to convey that each functional block performs one or more functions.

The functional decomposition process can have as many iterations as you need in order to achieve relevant granularity of the problem domain definition. As you can see in the following figure, functions at each layer of detail can be decomposed into even more detailed behavior. Functional blocks are accordingly decomposed into more elementary structures. It's important to mention that the granularity of system behavior and structure must be consistent in each level of detail.



After logical subsystems are identified, interactions between them can be specified. Every subsystem can have its own numerical characteristics, i.e., measurements of effectiveness. The functions of the SoI, its logical architecture, and MoEs all together establish the basis for specifying system requirements (see S1 cell).

Modeling the problem domain from the white-box perspective is explained step by step in related pages.

<table>
<tr><td></td><td colspan="6">PILLAR</td></tr>
<tr><td rowspan="7">DOMAIN</td><td colspan="2"></td><td>Requirements</td><td>Behavior</td><td>Structure</td><td>Parameters</td><td rowspan="7">Specialty Engineering

Integrated Testing

Analysis</td></tr>
<tr><td rowspan="2">Problem</td><td>Black Box</td><td rowspan="2">B1-W1 Stakeholder Needs</td><td>B2 Use Cases</td><td>B3 System Context</td><td>B4 Measurements of Effectiveness</td></tr>
<tr><td>White Box</td><td>W2 Functional Analysis</td><td>W3 Logical Subsystems Communication</td><td>W4 MoEs for Subsystems</td></tr>
<tr><td colspan="2" rowspan="4">Solution</td><td>S1 System Requirements</td><td>S2 System Behavior</td><td>S3 System Structure</td><td>S4 System Parameters</td></tr>
<tr><td>SS1 Subsystem Requirements</td><td>SS2 Subsystem Behavior</td><td>SS3 Subsystem Structure</td><td>SS4 Subsystem Parameters</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td></tr>
<tr><td>C1 Component Requirements</td><td>C2 Component Behavior</td><td>C3 Component Structure</td><td>C4 Component Parameters</td></tr>
<tr><td colspan="2">Implementation</td><td>I1 Physical Requirements</td><td colspan="3">Software, Electrical, Mechanical</td></tr>
</table>

W2. Functional Analysis: initial

What is it?

In this cell, you should go into deeper analysis of the system behavior by decomposing every function performed by the SoI in B2.

Modeling in this cell includes two phases: initial and final. During the initial phase of W2, top-level functions performed by the SoI are decomposed to specify more detailed system behavior. This helps to identify the functional blocks, also known as logical subsystems, that perform these functions. To capture these subsystems, you should switch to W3. Afterwards, you should return to W2 and specify which subsystems are responsible for performing which functions.

It is important to note that going into deeper analysis, the subfunctions of a decomposed function can be decomposed as well. Decomposition can be performed as many times as you need to achieve the relevant granularity of the problem you need to address. Every decomposed subfunction is regarded as a function from the standpoint of its subfunctions.

The initial phase of the W2 cell produces decomposed use case scenarios, also known as white-box scenarios.

Who is responsible?

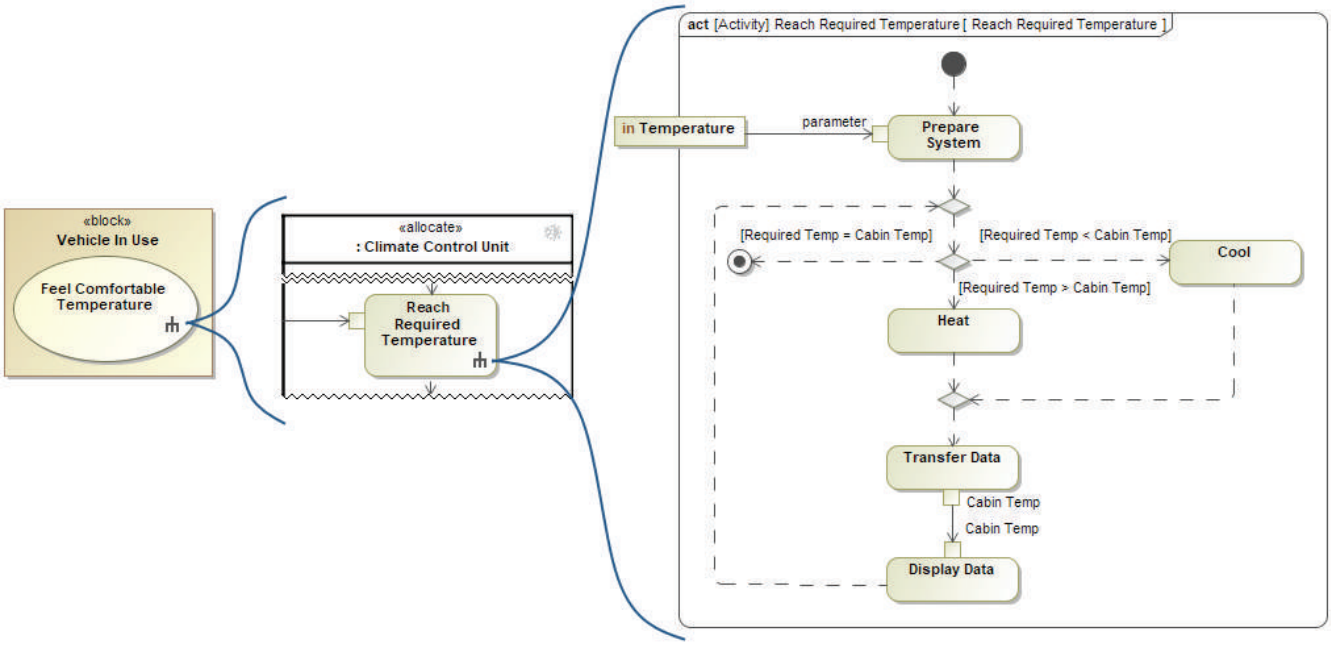
In a typical multidisciplinary systems engineering team, the functional analysis is performed by the *Requirements Team*.

How to model?

Functional analysis is a continuity of use case scenario refinements by using SysML activity diagrams.

A new SysML activity diagram should be created for every function in B2 allocated to SoI. Though there are two or even more swimlane partitions, you should choose only those functions that are nested under the partition that represents the block which captures your SoI. The following figure shows the white-box scenario of the *Reach Required Temperature* function (which is a part of the *Feel Comfortable Temperature* use case scenario) performed by the Climate Control Unit SoI.

Modeling the action flow of the white-box scenario stimulates the identification of logical subsystems of the SoI.



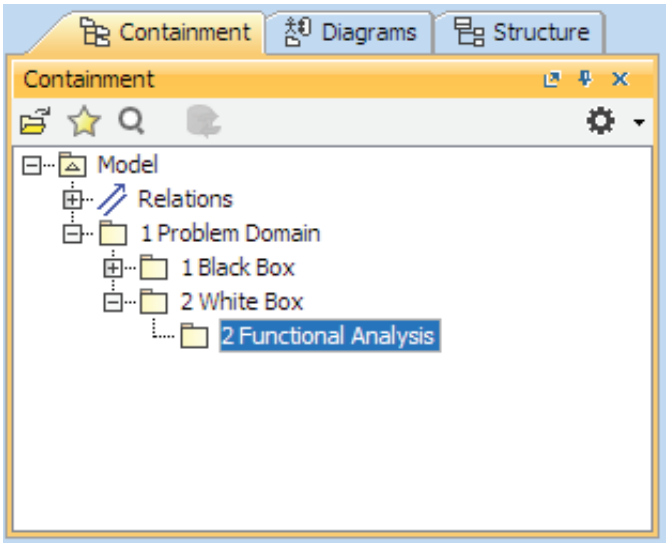
What's next?

- White-box scenarios stimulates the identification of logical subsystems of the SoI. Hence, the next cell is W3.

Tutorial

Step 1. Organizing the model for W2

According to the design of the MagicGrid framework, model artifacts that capture decomposed system functions should be stored under the structure of packages displayed in the following figure. As you can see, the top-level package represents the domain, the medium-level package represents the perspective, and the bottom-level package represents the cell.



To organize the model for W2

1. Right-click the *1 Problem Domain* package and select **Create Element**.
2. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.

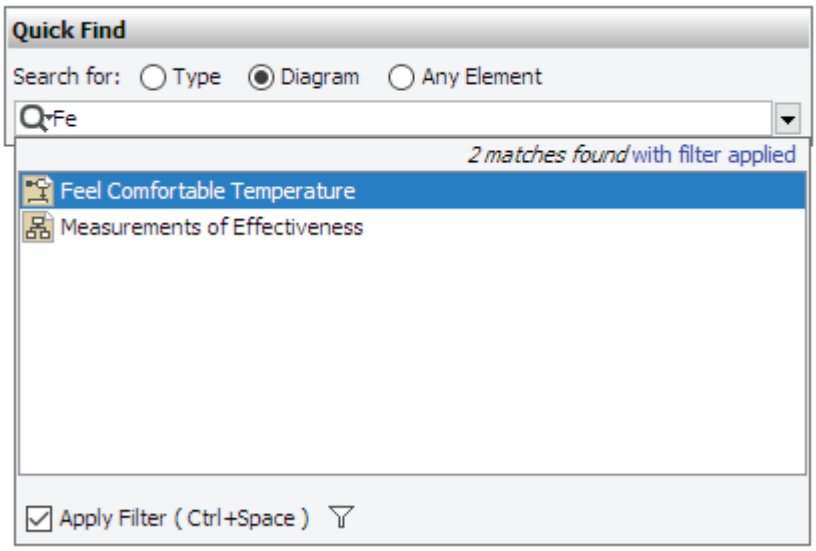
3. Type *2 White Box* to specify the name of the new package and press Enter.
4. Right-click the *2 White Box* package and select **Create Element**.
5. Repeat steps 2 and 3 to create the package named *2 Functional Analysis.*

Step 2. Creating an activity diagram to decompose a function

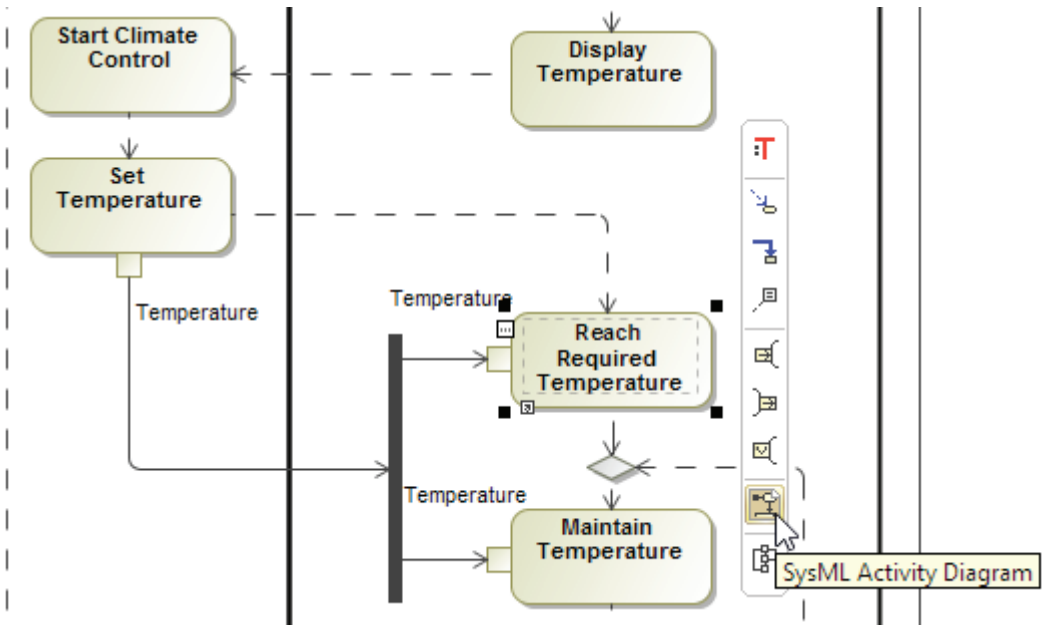
Let's say you want to decompose the function captured in your model as the *Reach Required Temperature* activity. If so, you need to create a SysML activity diagram inside that activity.

To create a SysML activity diagram inside the *Reach Required Temperature* activity

1. Open the *Feel Comfortable Temperature* activity diagram:
   a. Press Ctrl + Alt + F to open the **Quick Find** dialog.
   b. Click **Diagram** to search for diagrams only and type *Fe*.
   c. When you see the *Feel Comfortable Temperature* activity diagram selected in the search results list (see the following figure), press Enter. The diagram is opened.



2. Select the shape of the *Reach Required Temperature* action and click the SysML Activity Diagram button on its smart manipulator toolbar (see the following figure). A newly created activity diagram opens.

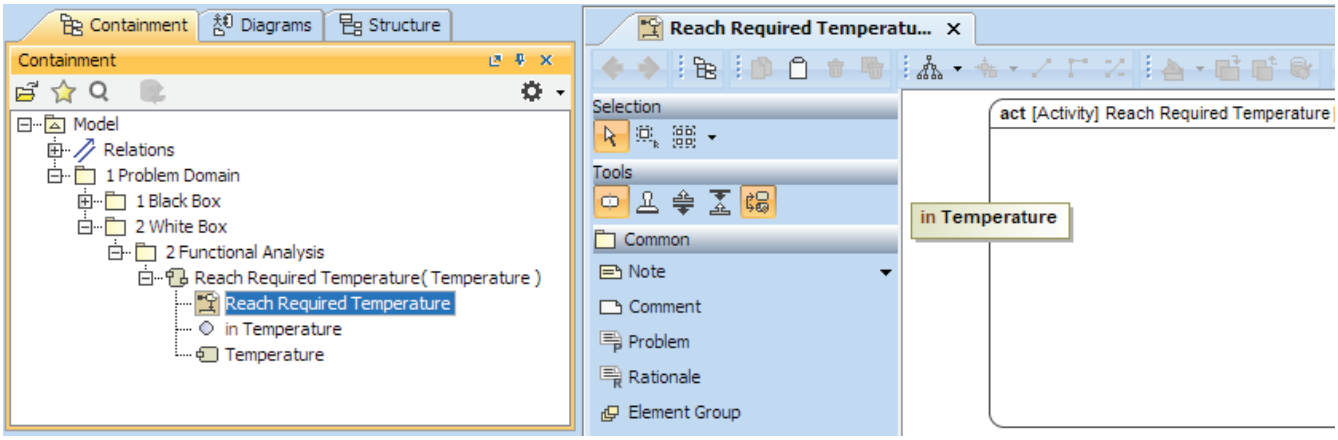


3. In the Model Browser, select the *Reach Required Temperature* activity:

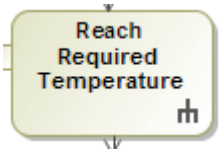
   a. Right-click the shape of the *Reach Required Temperature* action on the diagram pane.

   b. Select **Go To** > **Behavior Reach Required Temperature**. The activity is selected in the Model Browser.

4. Drag the *Reach Required Temperature* activity to the package *2 Functional Analysis*. The related activity diagram is moved in together.



ⓘ Now if you go back to the activity diagram *Feel Comfortable Temperature* (simply by clicking on the open diagram toolbar), you can see that the shape of the *Reach Required Temperature* action is decorated with the rake ( ) icon. The decoration means that the action contains an internal diagram, which opens simply by double-clicking the shape of that action.



Step 3. Specifying the white-box scenario

Now let's specify the white-box scenario of the function captured as the *Reach Required Temperature* activity.

Let's say the scenario includes the following steps:

1. Prepare System
2. Heat (if the required temperature *is higher* (>) than the temperature in the cabin)
3. Cool (if the required temperature *is lower* (<) than the temperature in the cabin)
4. Transfer Data
5. Display Data

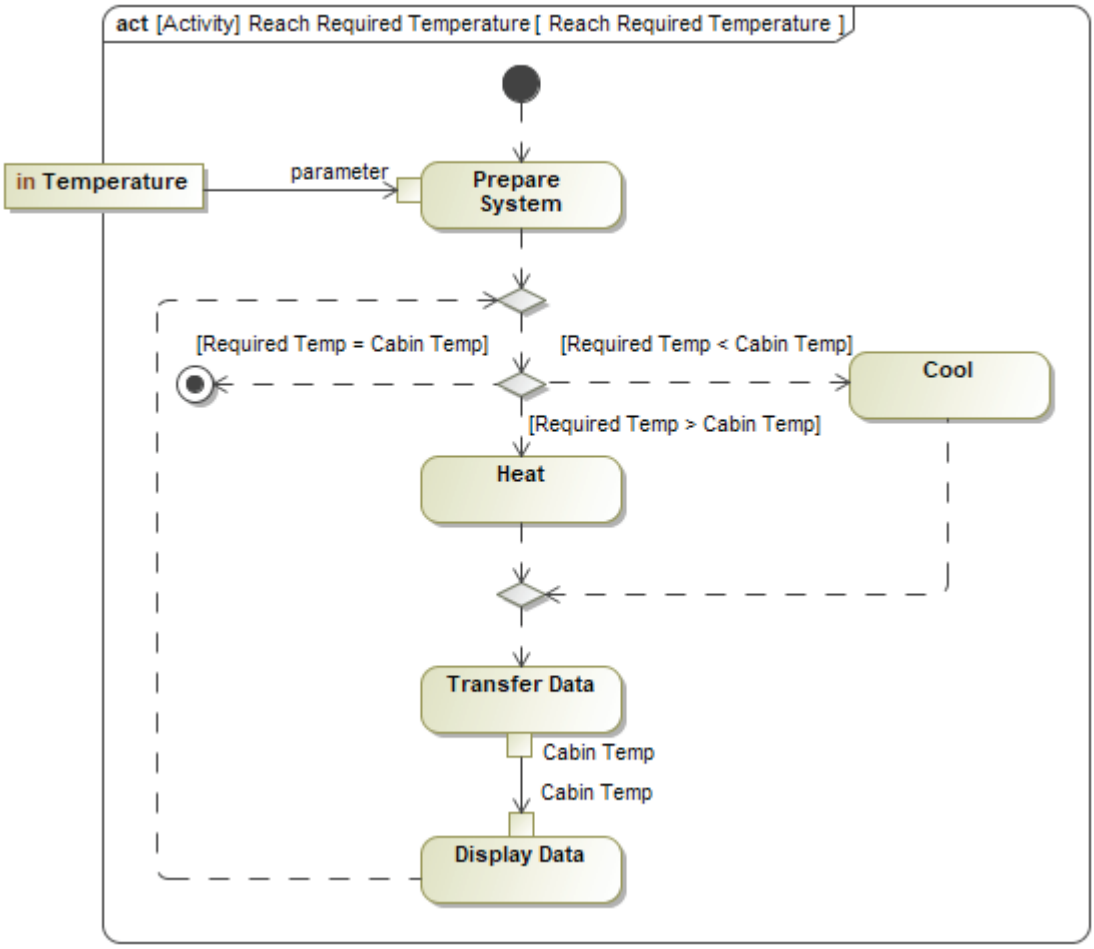
To specify the white-box scenario of the *Reach Required Temperature* activity

1. Open the *Reach Required Temperature* activity diagram, if not opened yet.
2. Be sure the Automatic Behavior Creation mode is enabled in the diagram. Otherwise, actions created in the diagram will not be typed by activities. For this, click the Automatic Behavior Creation button once or twice.
3. Click the **Initial Node** button on the diagram palette and then click an empty place on the diagram pane. The initial node is created.
4. Click the Control Flow button on the smart manipulator of the initial node shape, and then click an empty place on the diagram pane. A new action typed by the new activity is created.
5. Type *Prepare System* and press Enter.

   ⓘ Be sure you're typing the name after the colon (":"). Otherwise, the name is given to the action, but not to the activity which types that action.

6. Capture other items to build the complete scenario, which is displayed in the following figure. Here are the guidelines to help you:
   - To capture actions, use the smart manipulator toolbar, as in step 4.
   - To draw a control flow between existing elements, click the Control Flow button on the smart manipulator of the element to be at the tail end and then click the element to be at the arrow end.
   - To change the type of the element you want to create at the arrow end of the new control flow (for example, from action to decision or from decision to merge), right-click (instead of simply clicking!) the empty place in the diagram pane and select that type.

- To specify a guard, select the control flow and type the square bracket, and then specify the condition, for example, *[Required Temp > Cabin Temp.*



The scenario just created enables us to presume that the Vehicle Climate Control Unit may consist of four subsystems: one for control ling other subsystems, one for heating the air, and one for cooling it, and one for displaying data to the user. Thus, it's now time to move to the W3 cell and learn how to capture them.

W3. Logical Subsystems Communication

What is it?

While functional analysis within the initial phase of the W2 cell helps to identify logical subsystems, this cell is to define them in the model. The logical subsystem should be considered as a group of interconnected and interactive parts that performs one or more functions of the system of interest.

It's important to note that every subsystem can be decomposed into a more elementary structure. The number of iterations of such decomposition depends on the granularity level of the system architecture you want to address. Every subsystem is regarded as a system from the standpoint of its internal parts.

Overall, this cell produces:

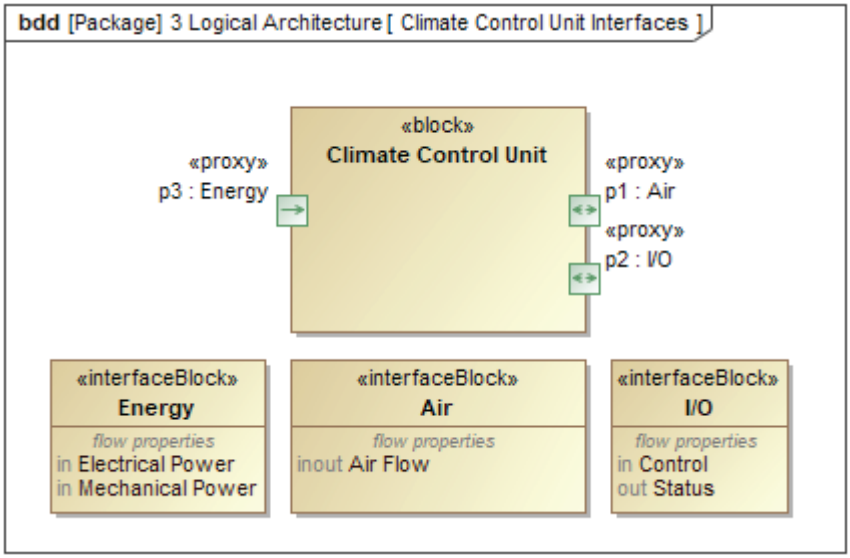
- Inputs and outputs of the SoI
- Definitions of logical subsystems of the SoI Interactions:
  - Between logical subsystems
  - Between logical subsystems and the outside of the system

Who is responsible?

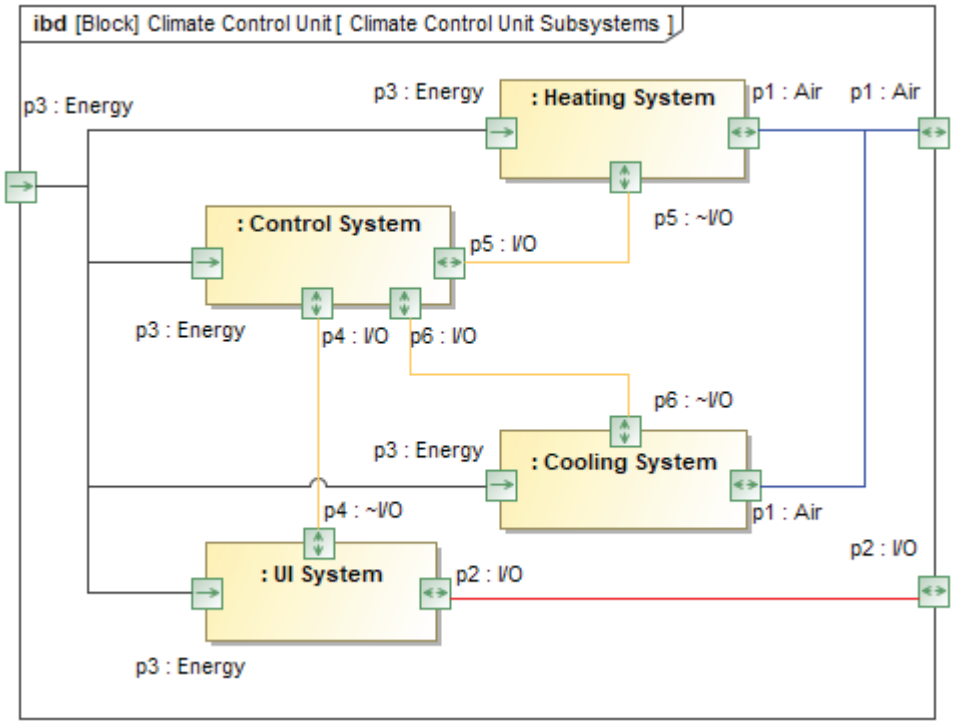
In a typical multidisciplinary systems engineering team, the structural decomposition of the system is performed by the *Requirements Team*.

How to model?

To define inputs and outputs of the SoI, you can use the infrastructure of the bdd created for the block, which represents that system in your model. These inputs and outputs can be specified as proxy ports of the system block. Some of them are usually identified by analyzing the system context specified in B3, while others are determined considering the results of the functional analysis performed in the initial phase of the W2 cell. Proxy ports should be typed by interface blocks, which we recommend storing in a separate folder to keep the model well-organized. The following figure shows the inputs and outputs of the Climate Control Unit.



To define the logical subsystems of the SoI and the interactions between them, you can utilize the infrastructure of the ibd created for the block that represents the SoI in your model. Logical subsystems, identified while performing the functional analysis in the initial phase of the W2 cell, can be specified as part properties of the block that represents the SoI. Blocks that type these part properties should be stored in a separate folder to keep the model well-organized. Connectors between these part properties indicate interactions between appropriate logical subsystems. Connectors between part properties and the diagram frame represent interactions between part properties and the outside of the system (inputs and outputs). Connections should be established via proxy ports. The following figure shows the logical subsystems of the Climate Control Unit.



If the functional analysis requires decomposing one or more subsystems of the SoI, we highly recommend using the same model structure pattern for each subsystem as you used for the SoI. Repeating the structure created at the highest level of abstraction for every lower level helps to establish the good and easily readable recursive structure of the model.

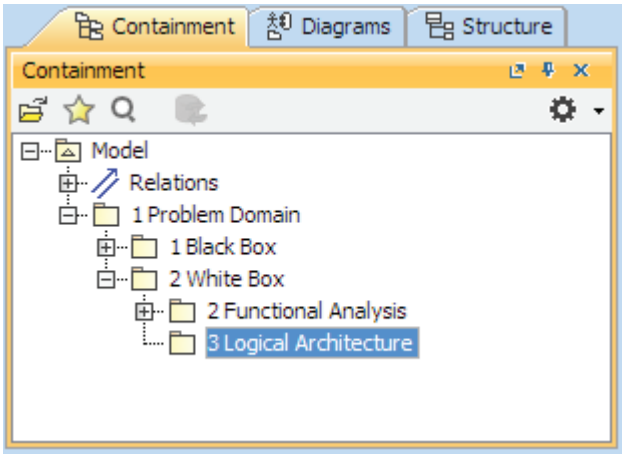
What's next?

- After the logical subsystems are defined and interactions between them are specified, you can complete the functional analysis by grouping the subfunctions by logical subsystems, which means you should move to W2.final.

Tutorial

Step 1. Organizing the model for W3

Following the structure of the MagicGrid framework, model artifacts that capture system architecture should be stored in a separate package inside the *2 White Box package*. We recommend naming the package *Logical Architecture*.



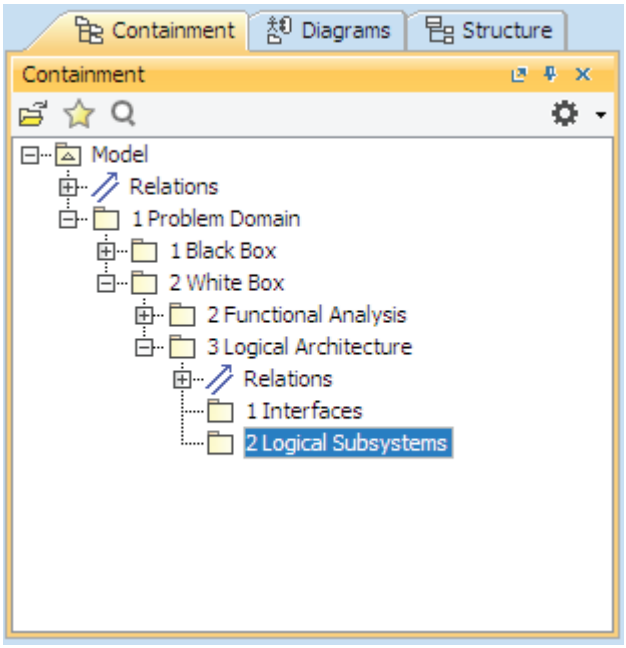
To organize the model for W3

1. Right-click the *2 White Box* package and select **Create Element**.
2. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.
3. Type *3 Logical Architecture* to specify the name of the new package and press Enter.

For the purpose of keeping the inner structure of the *3 Logical Architecture* package well-organized, you need to create a few more packages. These are:

- *1 Interfaces*
- *2 Logical Subsystems*

When you create them on your own (by following the previous procedure), the Model Browser of your solution domain model should look the same as displayed in the following figure.



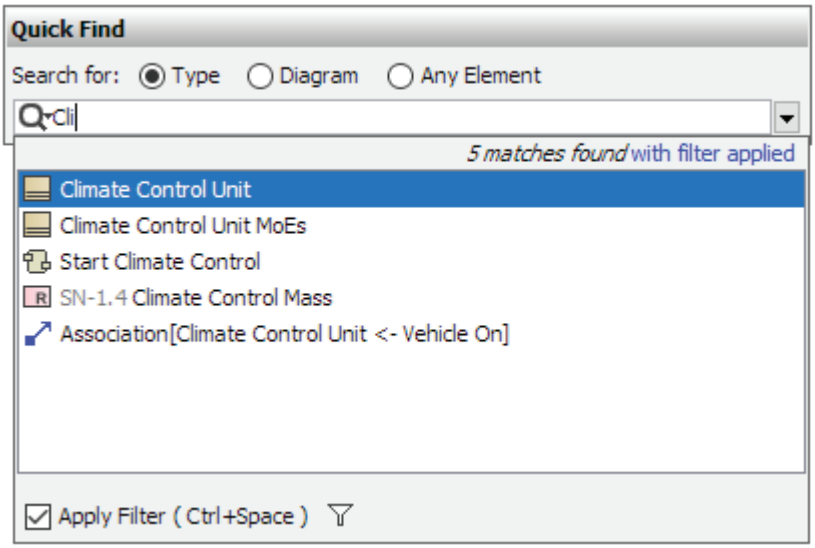
Step 2. Creating a bdd for capturing SoI interfaces

To start capturing interfaces, which specify inputs and outputs of the SoI, you first need to create a bdd and display the block that captures the system on the diagram pane. But before this, you should change the location of that block in your model by moving it to the package *3 Logical Architecture.*

To create and prepare a bdd for capturing SoI interfaces

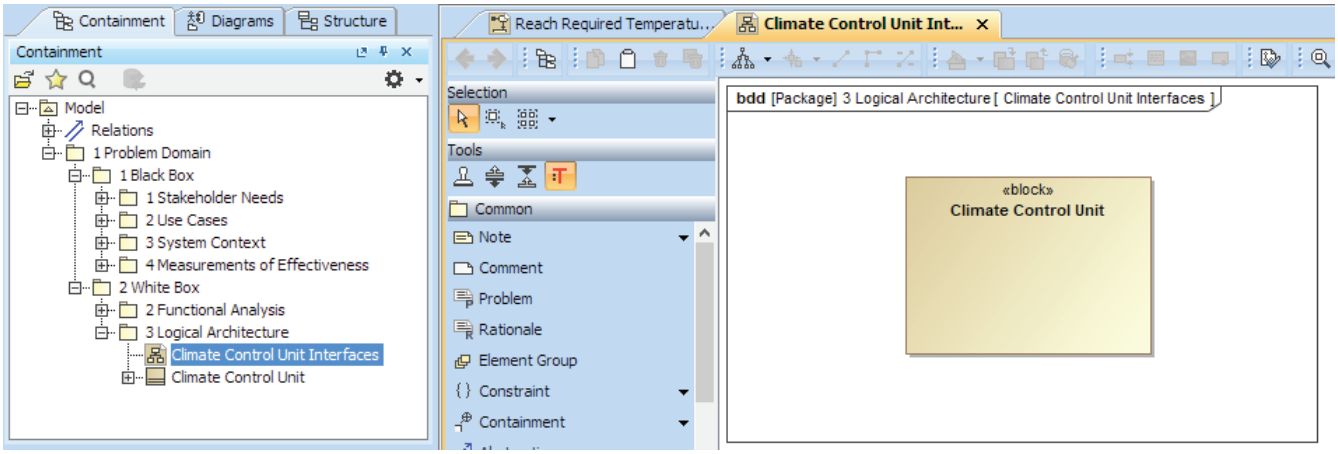
1. Create the bdd:
   a. Right-click the *3 Logical Architecture* package and select **Create Diagram**.
   b. In the search box, type *bdd*, the acronym of the SysML block definition diagram, and then press Enter. The diagram is created.
   c. Type *Climate Control Unit Interfaces* to specify the name of the new diagram and press Enter again.
2. In the Model Browser, select the *Climate Control Unit* block. For this, use the quick find capability:
   a. Press Ctrl + Alt + F. The **Quick Find** dialog opens.
   b. Type *Cli*.

c. When you see the *Climate Control Unit* block selected in the search results list (see the following figure), press Enter. The *Climate Control Unit* block is selected in the Model Browser.



3. Drag the *Climate Control Unit* block to the newly created package *3 Logical Architecture* (see step 1 of the W3 tutorial).

4. Drag the *Climate Control Unit* block to the diagram pane. The shape of the block appears on the diagram.



Step 3. Capturing SoI interfaces

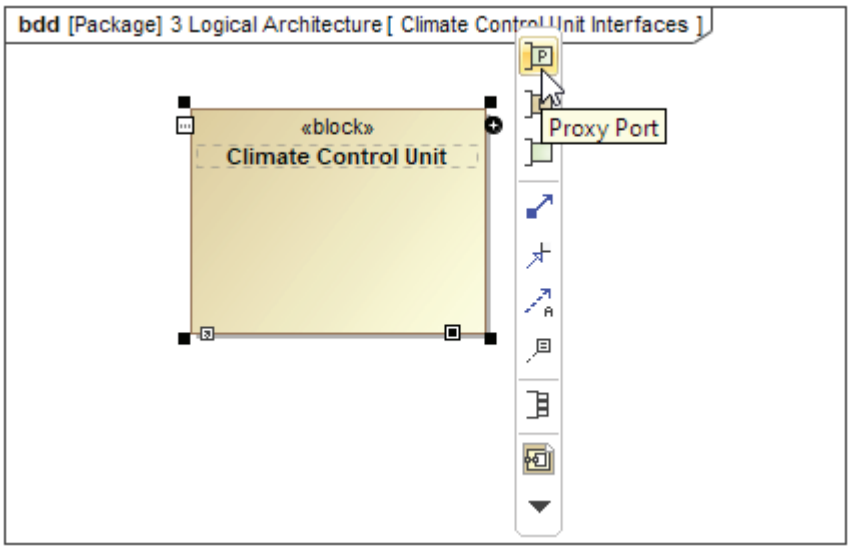
By analyzing the system context specified in the final phase of the B3 cell, we can conclude that the Climate Control Unit takes air from the cabin of the vehicle and must be controlled by the vehicle occupant. These are inputs of the SoI. It also requires electrical and mechanical power, as can be derived from the results of the functional analysis performed in the initial phase of the W2 cell.

The functional analysis results also reveal that the Climate Control Unit provides cooled or heated air to the cabin. System status, like *Climate Control On* or *Climate Control Off* and the temperature of the air in the cabin, is output too. This can be identified from the system context specified in the final phase of the B3 cell.

Inputs and outputs can be specified as proxy ports of the *Climate Control Unit* block. Proxy ports don't need names, but they should be typed by interface blocks, which define the above described inputs and outputs. Flow properties of the interface block with *in* direction indicate input, and flow properties with *out* direction indicate output.

To capture system interfaces

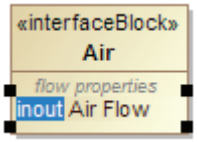
1. Open the diagram *Climate Control Unit Interfaces*, if not opened yet.
2. Make sure the Type Selection Mode is on in the diagram. This mode enables you to type proxy ports (with interface blocks) instantly after they are created.
3. Select the shape of the *Climate Control Unit* block and on its smart manipulator click the Proxy Port button (see the following figure). A new proxy port appears on the border of the block shape.



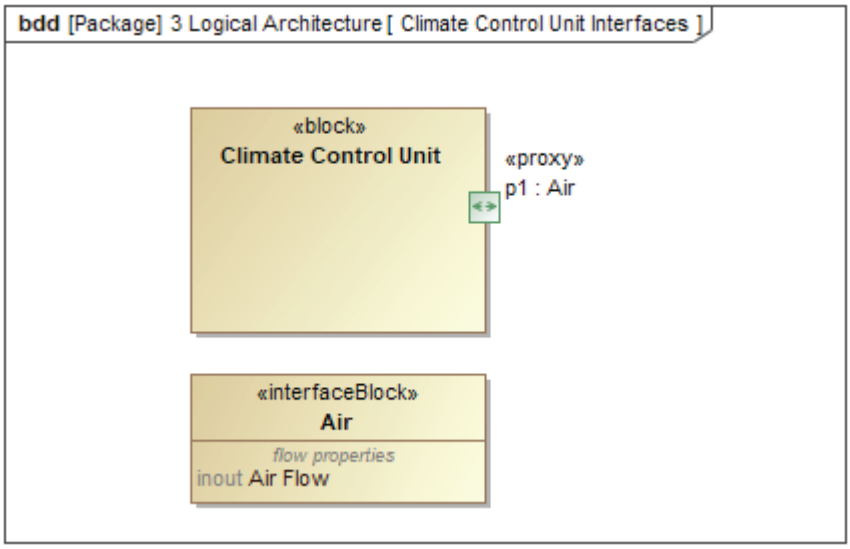
ⓘ Proxy ports are by default named *p1, p2, ..., pn*. It's not necessary to rename them, since they don't convey any semantics.

4. Type *Air* next to the colon (":") directly on the proxy port shape and press Enter. A new interface block to type the proxy port is created and displayed in the Model Browser, within the *3 Logical Architecture* package.
5. Select that interface block and drag it to the diagram pane. A shape of the interface block appears on the diagram.
6. Click the Create Element button on that shape and select **Flow Property**.
7. Type *Air Flow* directly on the shape of the interface block to specify the name of the new flow property and press Enter.

ⓘ In this case, you don't need to modify the direction of the flow property, since the default direction is inout as it should be. In the opposite case, you can change the direction simply by removing the irrelevant part of the direction indicator (*in* or *out*).

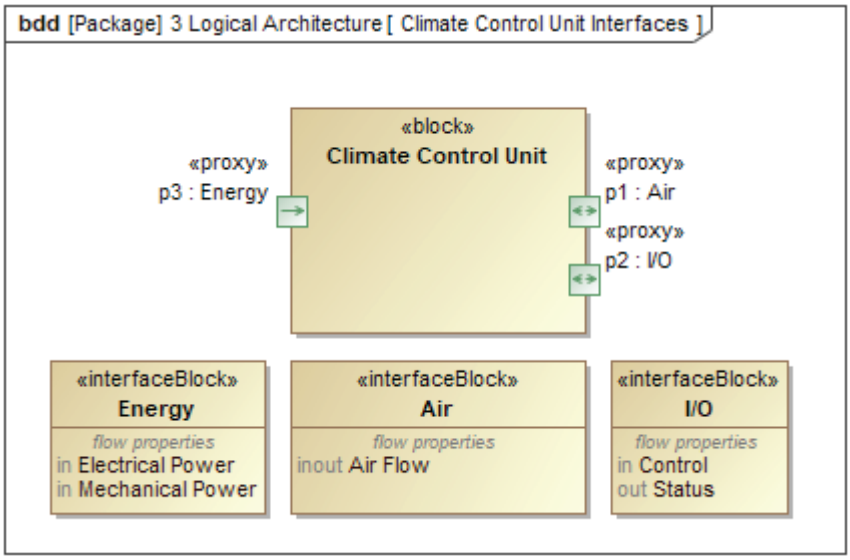


As a result, a bidirectional arrow appears on the shape of the proxy port (see the following figure). This indicates that the proxy port is both the input and output interface of your SoI.



8. Repeat step 3 to step 7 to specify other interfaces. After you're done, the contents of the diagram Climate Control Unit Interfaces should look similar to the one in the following figure.

ⓘ We recommend placing shapes of *in* proxy ports on the left border of the *Climate Control Unit* block, and *out* and *inout* proxy ports on the right border.



ⓘ The following figure displays the Blackbox ICD Table, an alternative view of the *Climate Control Unit interfaces*. This tutorial doesn't explain how to build the table, but you can find this information in the latest documentation of the modeling tool.

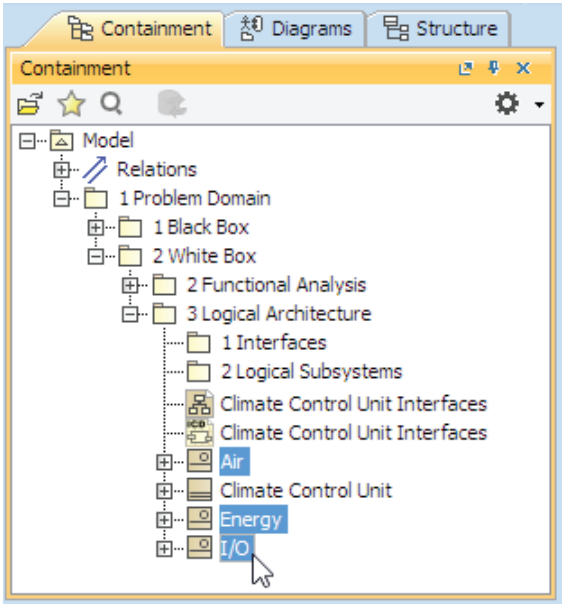
Criteria

Element Type: Port ... Block: Climate Cont ...

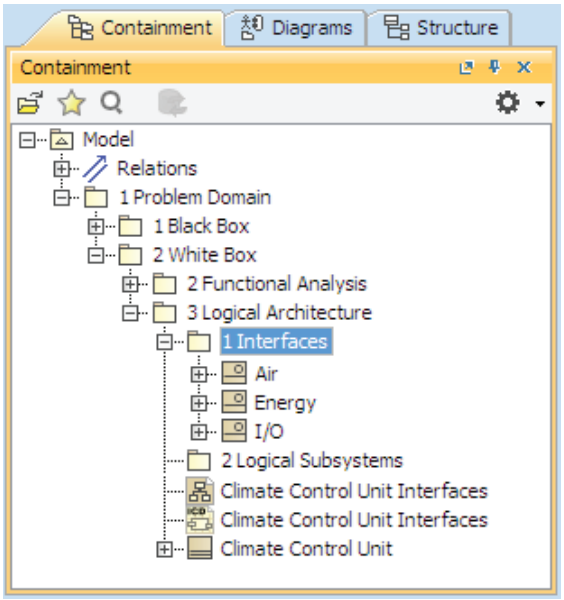
#	Port Name	Port Type	Type Features	Direction
1	p3	Energy	in Electrical Power in Mechanical Power	in
2	p1	Air	inout Air Flow	inout
3	p2	I/O	in Control out Status	inout

9. In the Model Browser, select all the interface blocks and drag them to the *1 Interfaces* package you've created in step 1 of the W3 tutorial.

ⓘ To select the set of non-adjacent items in the tree, click the first one, press Ctrl, and while holding it down, select other items one by one.



The blocks are moved to the *1 Interfaces* package.



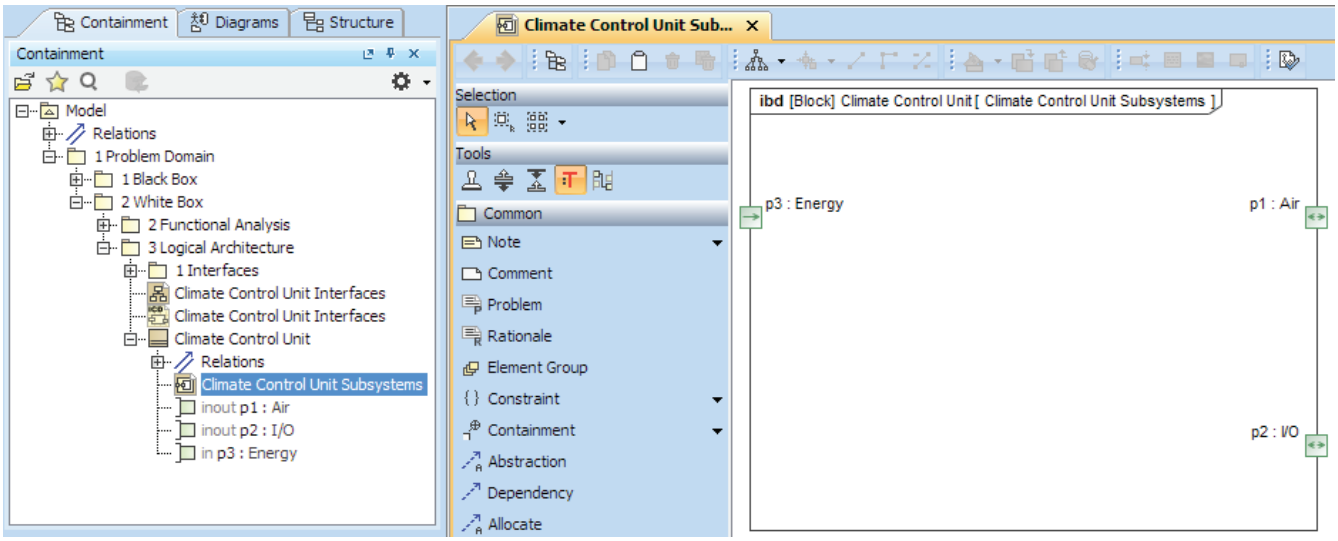
Step 4. Creating an ibd for capturing logical subsystems

Logical subsystems of the Climate Control Unit can be specified in an ibd, which is owned by the *Climate Control Unit* block. According to the SysML, the frame of this diagram represents the borders of the SoI and thus allows you to specify interactions between the internal parts of the SoI and the outside of the system via proxy ports (see step 3 of the W3 tutorial), displayed on that frame.

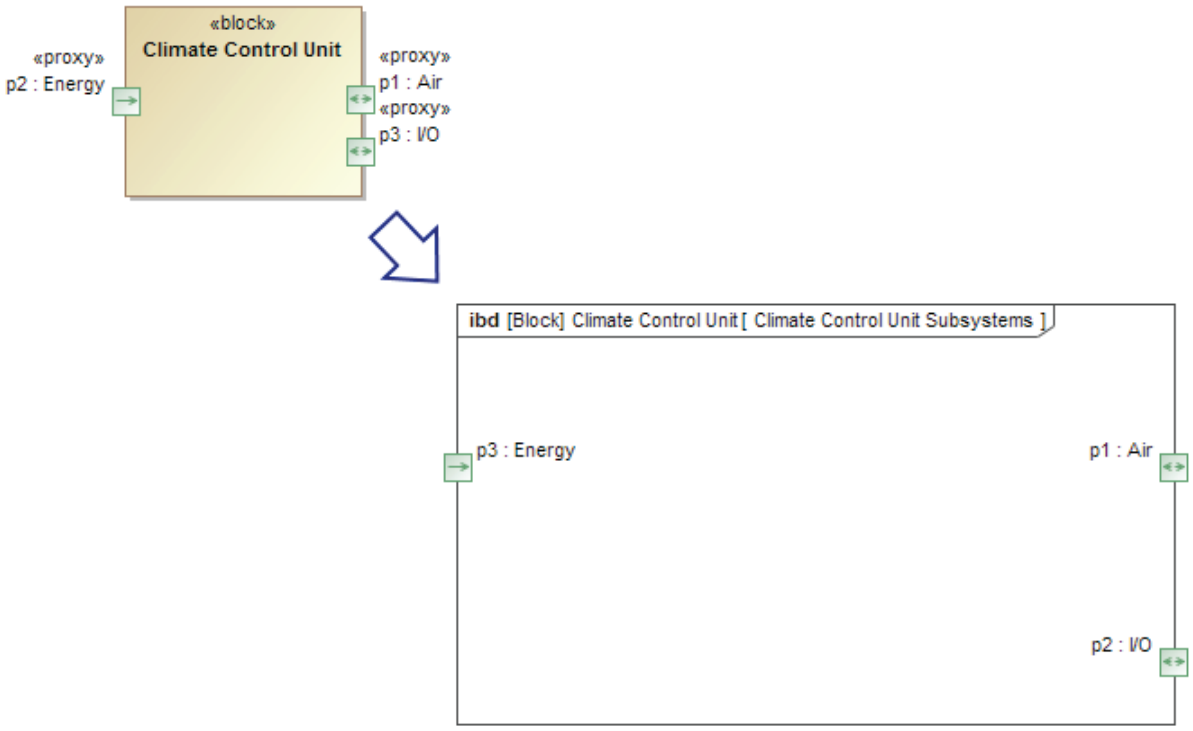
To create an ibd for capturing logical subsystems of th e SoI

1. In the Model Browser, right-click the *Climate Control Unit* block and select **Create Diagram**.
2. In the search box, type *ibd*, the acronym of the SysML internal block diagram, and then press Enter. The **Display Parts/Ports** dialog opens.
3. Click **Clear All** and then click **Proxy Port** on the right side of the dialog. Only proxy ports of the *Climate Control Unit* block become selected in the tree on the left.
4. Click **OK**. The diagram is created, and the proxy ports are displayed on its frame.
5. Type *Climate Control Unit Subsystems* to specify the name of the new diagram and press Enter again.
6. Move the shapes of the in proxy ports onto the left border of the diagram frame, and the shapes of the inout proxy ports onto the right.

The final view of your diagram should be similar to the one in the following figure.



ⓘ While the Climate Control Unit Interfaces bdd (see step 3 of the W3 tutorial) represents the SoI as a black box, the idb is created to represent the same system as a white box: the *Climate Control Unit* block is opened to see its internal structure. That's why proxy ports displayed on the borders of the *Climate Control Unit* block in the bdd are displayed on the diagram frame in the ibd.



Step 5. Capturing logical subsystems

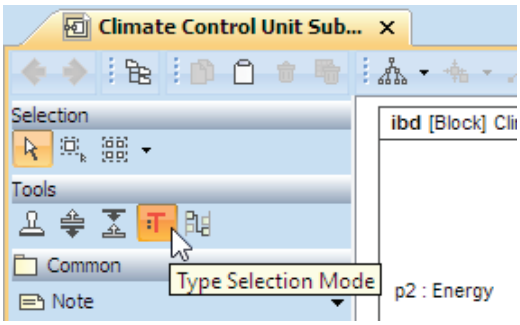
As the functional analysis in the initial phase of the W2 cell reveals, the Climate Control Unit consists of these logical subsystems:

- Control System
- User Interface (UI) System
- Heating System
- Cooling System

These subsystems can be specified as part properties of the *Climate Control Unit* block. Part properties don't need names, but they should be typed by blocks that define the above-declared logical subsystems.

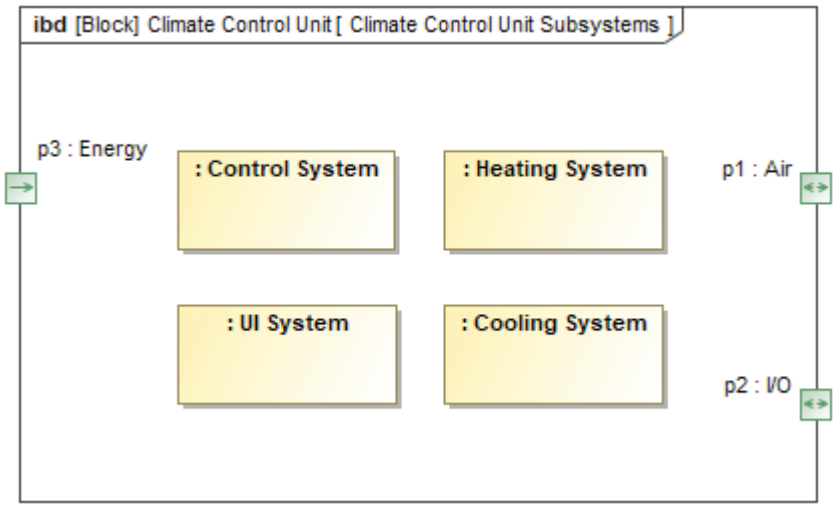
To define logical subsystems

1. Open the *Climate Control Unit Subsystems* ibd, created in step 5 of the W3 tutorial, if not yet opened.
2. Make sure the Type Selection Mode is on in the diagram. Otherwise, parts created in this diagram will not be typed by blocks.



3. Click the **Part Property** button on the diagram palette and then click an empty place on the diagram pane. An unnamed part property is created, and the list of existing blocks to type it is offered.
4. Type *Control System* next to the colon ("**:**") directly on the part property shape and press Enter. The *Control System* block to type the just-created part property is created in the Model Browser.
5. Repeat steps 3 and 4 to create another logical subsystems from the list above.

When you're done, your ibd Climate Control Unit Subsystems should look very similar to the one below.



Step 6. Specifying interactions in the context of the SoI

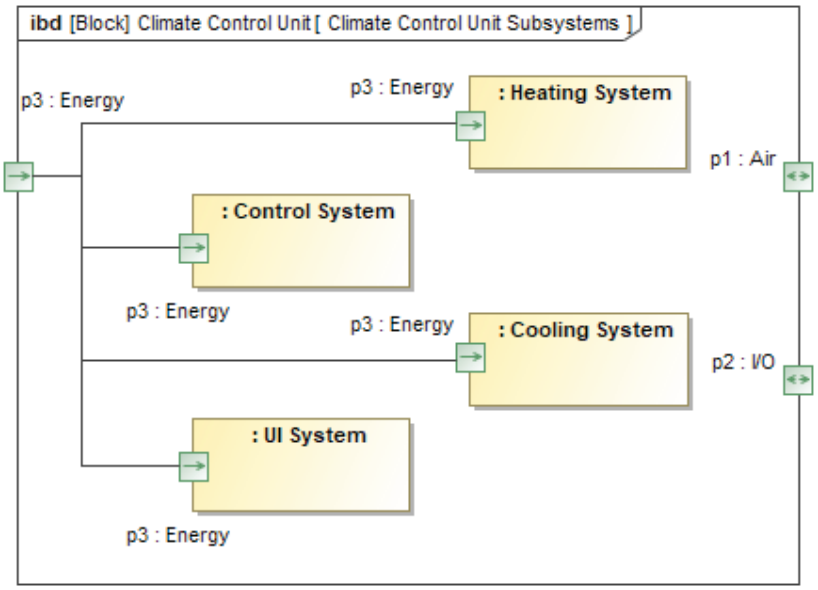
Logical subsystems can interact with each other as well as with the outside of the system. An interaction between two subsystems can be specified as a connector linking a couple of appropriate part properties. The one between the subsystem and the outside of the SoI can be specified as a connector linking the appropriate part property and the diagram frame (here you should remember that the diagram frame in the ibd represents the borders of the SoI). In both cases, connectors are established via proxy ports.

It's easier to start with specifying interconnections that come from the outside of the SoI. Let's specify that all logical subsystems of the Climate Control Unit require electrical or mechanical power; that is, energy in general.

To draw a connector between the diagram frame and the *Control System* part property via the *Energy* proxy port

1. Select the *Energy* proxy port and click the Connector button on its smart manipulator toolbar.
2. Click the *Control System* part property.
3. Select **New Proxy Port**. A proxy port is created on the *Control System* part property, and the connector is established between the diagram frame and the selected part property.

Repeat the procedure to draw the rest of the adequate connectors. When you're done, your *Climate Control Unit Subsystems* ibd should look very similar to the one below.

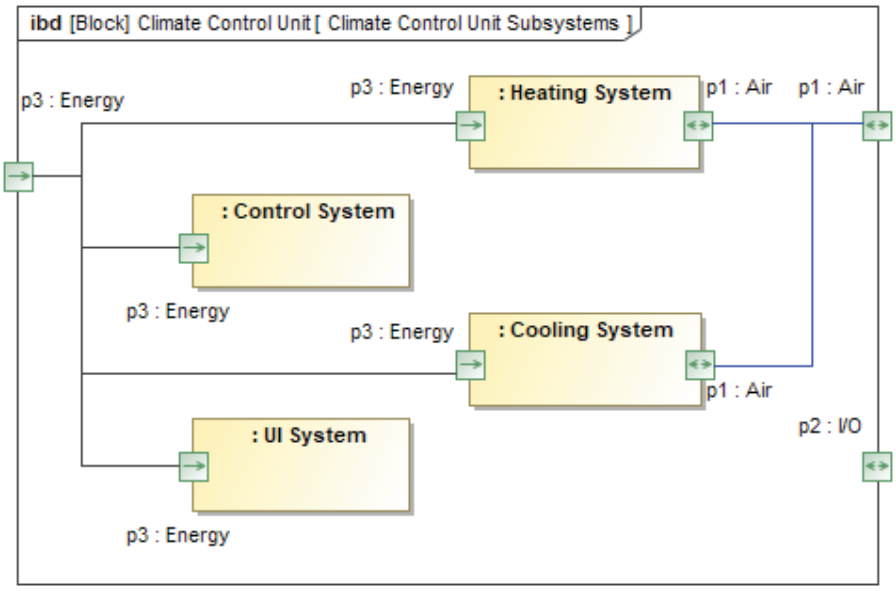


Also, it's necessary to specify that the Heating System takes some air from the cabin and provides the heated air back.

To draw a connector between the diagram frame and the *Heating System* part property via the *Air* proxy port

1. Select the *Air* proxy port and click the Connector button on its smart manipulator toolbar.
2. Click the *Heating System* part property.
3. Select **New Proxy Port**. A proxy port is created on the *Heating System* part property, and the connector is established between the diagram frame and the selected part property.

Repeat the procedure to draw another connector, this time between the diagram frame and the Cooling System part property to specify that the Cooling System takes some air from the cabin too, and provides the cooled air back. As a result, your diagram should look like the one in the following figure.

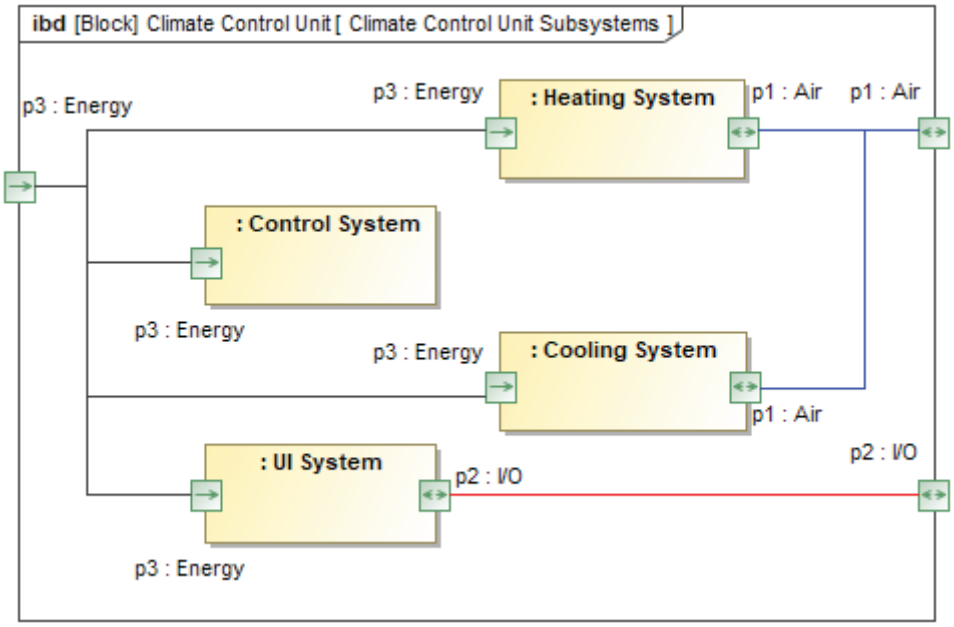


ⓘ The connector's color (as well as it's other features) can be changed by modifying the symbol properties of that connector. For this, right-click the connector and select **Symbol Properties**. In the open dialog, select the value cell of the **Pen Color** property and click the **...** button to open the **Color** dialog. Then select blue, or any other color you want, and close both dialogs.

Another interaction with the outside of the SoI is accepting commands from the user, that is, the vehicle occupant, and providing information, such as the status of the Climate Control Unit or cabin temperature.

To draw the connector between the diagram frame and the *UI System* part property via the *I/O* proxy port

1. Select the *I/O* proxy port and click the Connector button on its smart manipulator toolbar.
2. Click the *UI System* part property.
3. Select **New Proxy Port**. A proxy port is created on the *UI System* part property, and the connector is established between the diagram frame and the selected part property.

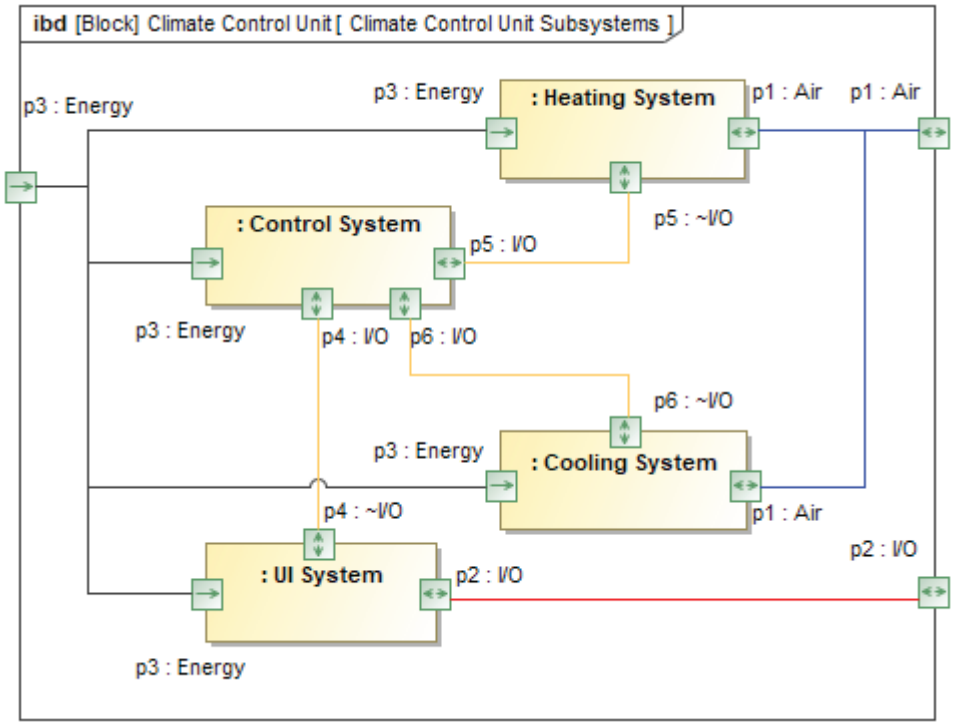


Now it's time to specify interactions between subsystems, so let's define that the Control System governs the rest of the logical subsystems in the Climate Control Unit and receives status from each of them.

To draw a connector between the *Control System* and *UI System* part properties

1. Select the *Control System* part property and click the Proxy Port button on its smart manipulator toolbar.
2. Select **I/O**. The new proxy port becomes typed by the *I/O* interface block.
3. Select the Connector button on its smart manipulator toolbar.
4. Click the *UI System* part property.
5. Select **New Proxy Port**. A proxy port is created on the UI System part property, and the new connector is created between these part properties.

Repeat the procedure to draw connectors from the *Control System* part property to the *Heating System* and Cooling System part properties. When you're done, your *Climate Control Unit Subsystems* ibd should look very similar to the one below.



ⓘ The following figure displays the alternative view of the *Climate Control Unit Subsystems* ibd. This tutorial doesn't explain how to build the table, but you can find this information in the latest documentation of the modeling tool.

Criteria

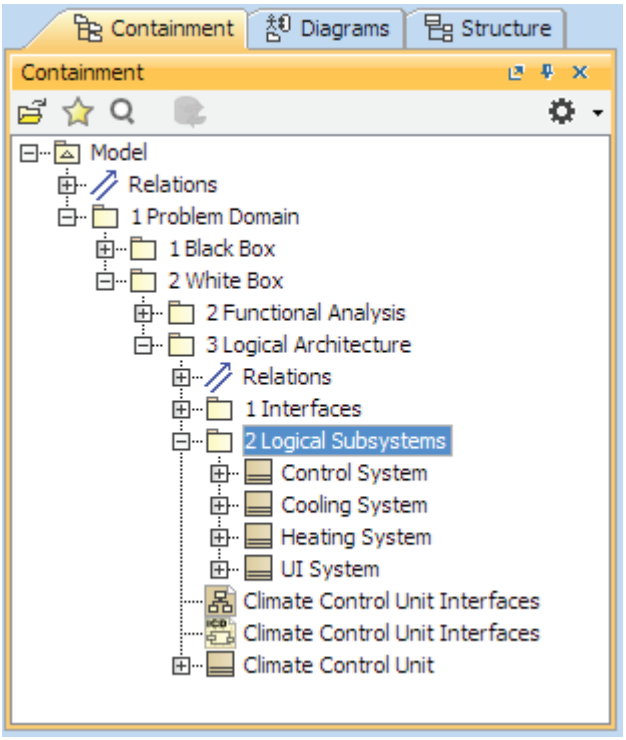
Element Type: Connector ... Context: Climate Control Unit ... Filter:

#	Part A	Port A	Port A Features	Port B	Port B Features	Part B
1	: Control System	inout p4 : I/O	in Control out Status	inout p4 : ~I/O	out Control in Status	: UI System
2	: Control System	inout p5 : I/O	in Control out Status	inout p5 : ~I/O	out Control in Status	: Heating System
3	: Control System	inout p6 : I/O	in Control out Status	inout p6 : ~I/O	out Control in Status	: Cooling System
4	Climate Control Unit	inout p1 : Air	inout Air Flow	inout p1 : Air	inout Air Flow	: Cooling System
5	Climate Control Unit	inout p1 : Air	inout Air Flow	inout p1 : Air	inout Air Flow	: Heating System
6	Climate Control Unit	inout p2 : I/O	in Control out Status	inout p2 : I/O	in Control out Status	: UI System
7	Climate Control Unit	in p3 : Energy	in Electrical Power in Mechanical Power	in p3 : Energy	in Electrical Power in Mechanical Power	: Control System
8	Climate Control Unit	in p3 : Energy	in Electrical Power in Mechanical Power	in p3 : Energy	in Electrical Power in Mechanical Power	: Cooling System
9	Climate Control Unit	in p3 : Energy	in Electrical Power in Mechanical Power	in p3 : Energy	in Electrical Power in Mechanical Power	: Heating System
10	Climate Control Unit	in p3 : Energy	in Electrical Power in Mechanical Power	in p3 : Energy	in Electrical Power in Mechanical Power	: UI System

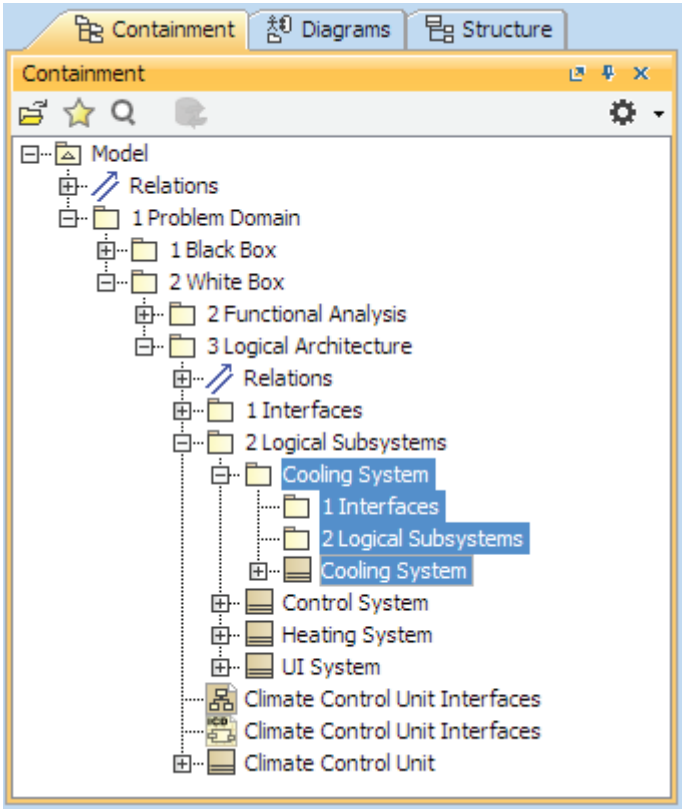
The blocks defining logical subsystems are by default stored in the same package with the *Climate Control Unit* block. To keep the model well-organized, we recommend moving them into the *2 Logical Subsystems* package, a subpackage of the *3 Logical Architecture* package.

To organize the blocks that capture logical subsystems into the *2 Logical Subsystems* package

1. In the Model Browser, select all the blocks that capture the logical subsystems.
2. Drag the selection to the *2 Logical Subsystems* package.



ⓘ When you decide to decompose these subsystems, remember that the model structure for specifying each subsystem should be created by using the same structure pattern as the one for the SoI. The figure below illustrates the structure of the system model in the case of decomposing the Cooling System.

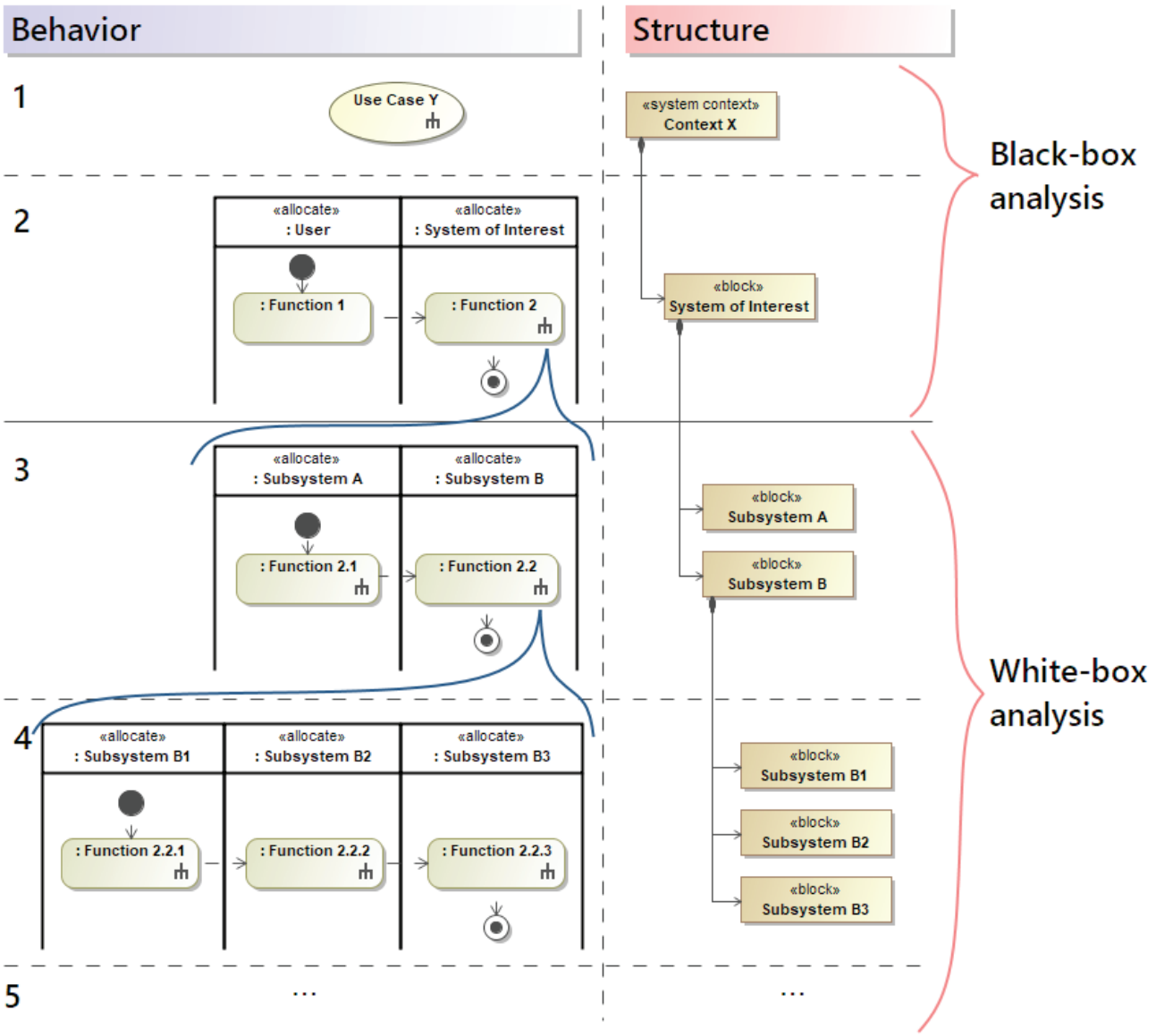


W2. Functional Analysis: final

What is it?

The final phase of the W2 cell is necessary to complete the functional analysis of the SoI. The completion requires allocating system behavior to the structure of that system. In other words, you should specify what logical subsystems defined in W3 are responsible for performing each function. For this, functions are grouped by logical subsystems.

Here it's important to understand that granularity of system behavior and structure must be consistent in each level of detail. This means you cannot assign the function of the subsystem component to that subsystem. That's why we recommend performing W2.initial, W3, and W2.final in precise sequence at each iteration of functional-structural decomposition of the SoI. Going into deeper analysis is only possible after you're completely done with the current level of granularity. The recommended correspondence between the granularity levels of behavioral and structural decomposition is displayed in the following figure.



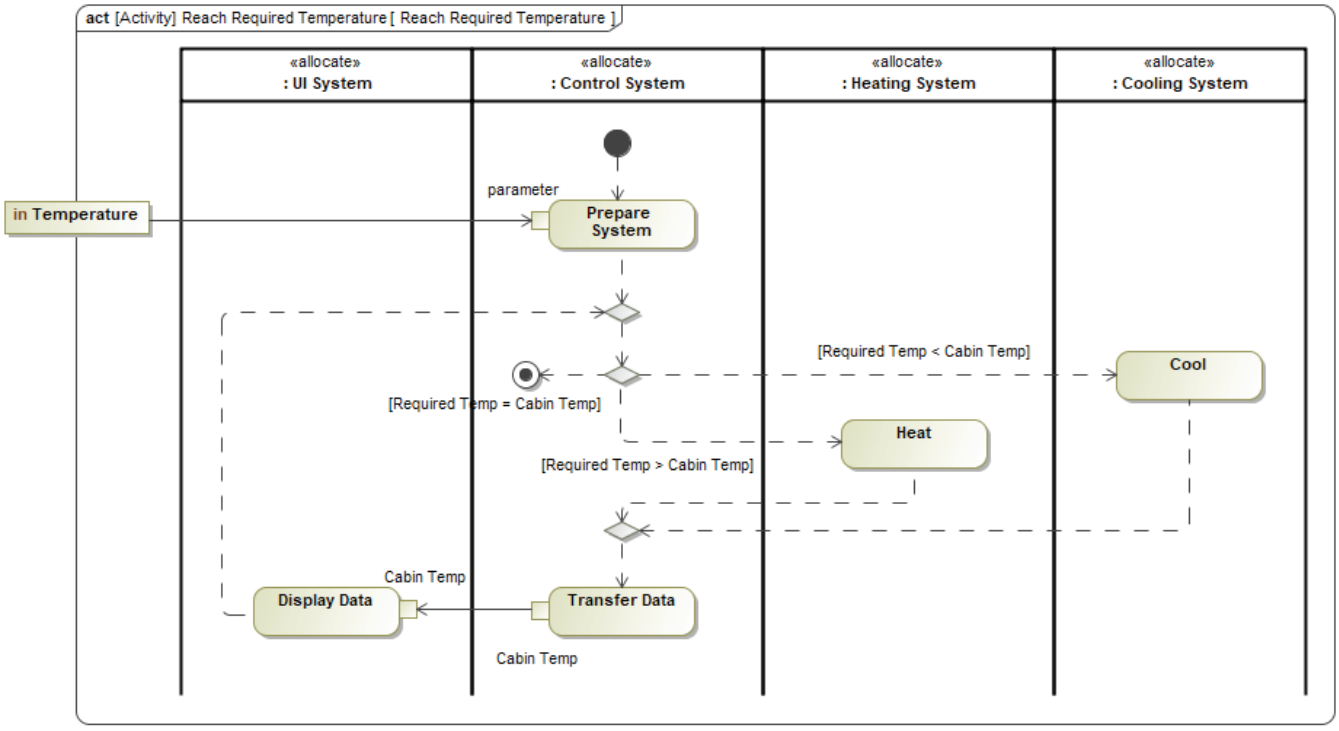
Who is responsible?

In a typical multidisciplinary systems engineering team, the functional analysis is performed by the *Requirements Team*.

How to model?

In the SysML activity diagram that displays the white-box scenario captured in the initial phase of the W2 cell, subsystems can be represented as swimlane partitions. A swimlane partition actually represents the part property of the SoI block. An action which captures a system function can be easily assigned to a relevant swimlane partition by dragging that action onto it.

The following figure displays the subfunctions of the Reach Required Temperature function, grouped by the logical subsystems of the SoI: *Control System*, *UI System, Heating System*, and *Cooling System*.



What's next?

- System functions and logical architecture, with or without numerical characteristics of the SoI, establish the basis for the system requirements specification. Thus, you can switch to W1.
- You might also need to specify numerical characteristics for one or more logical subsystems, which would be more specific than those defined in B4 for the entire SoI. In such case, you should move to W4.

Tutorial

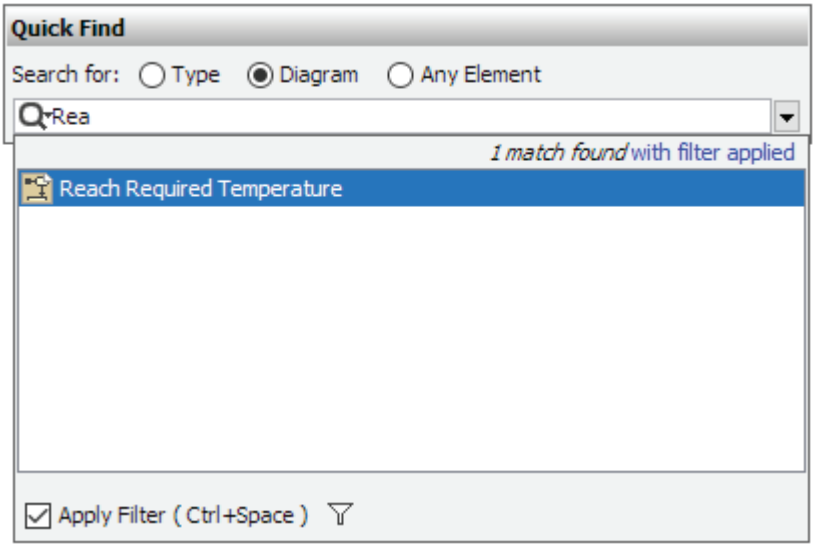
Step 1. Grouping functions by logical subsystems

Now it's time to update the SysML activity diagram *Reach Required Temperature* (see step 2 of the W initial phase tutorial) with swimlane partitions, which represent logical subsystems of the Vehicle Climate Control Unit. In sequel, actions should be allocated to these swimlane partitions in order to specify what logical subsystems are responsible for preforming each function.

To group functions by logical subsystems

1. Open the SysML activity diagram *Reach Required Temperature*:
   a. Press Ctrl + Alt + F to open the **Quick Find** dialog.
   b. Click **Diagram** to search for diagrams only and type *Rea*.

c. When you see the SysML activity diagram *Reach Required Temperature* selected in the search results list (see the following figure), press Enter. The diagram is opened.

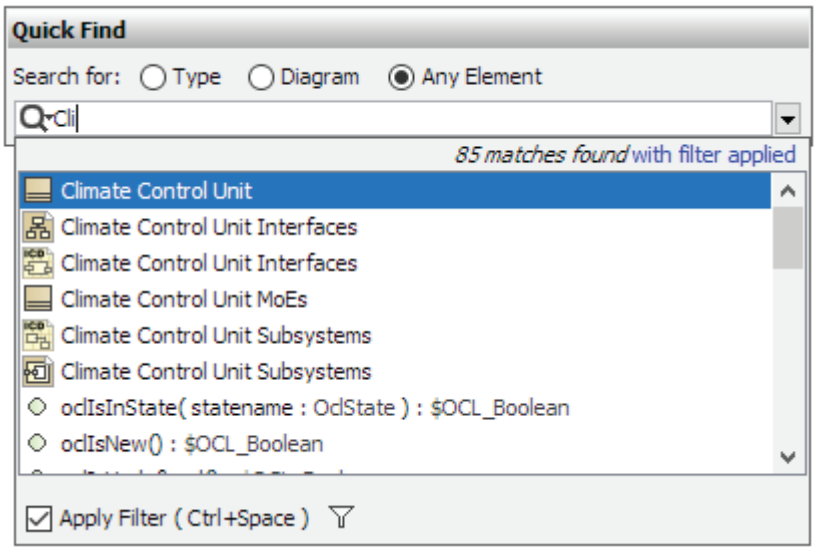


2. Select the *Climate Control Unit* block in the Model Browser:

a. Press Ctrl + Alt + F to open the **Quick Find** dialog.

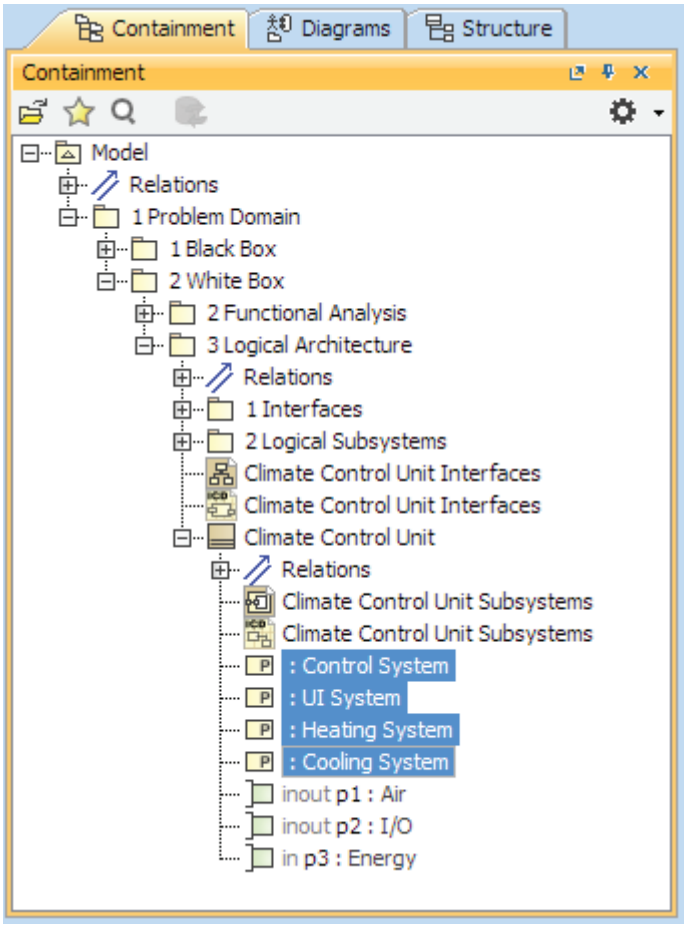
b. Click **Any Element** to search for particular elements only and type *Cli*.

c. When you see the *Climate Control Unit* block selected in the search results list (see the following figure), press Enter.



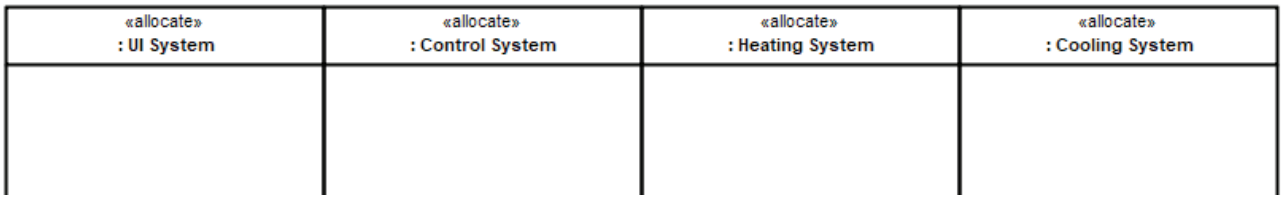
3. Expand the selected block, if not yet expanded, and select all the part properties it contains.

ⓘ To select the set of adjacent items in the tree, click the first one, press Shift, and while holding it down, select the last one.



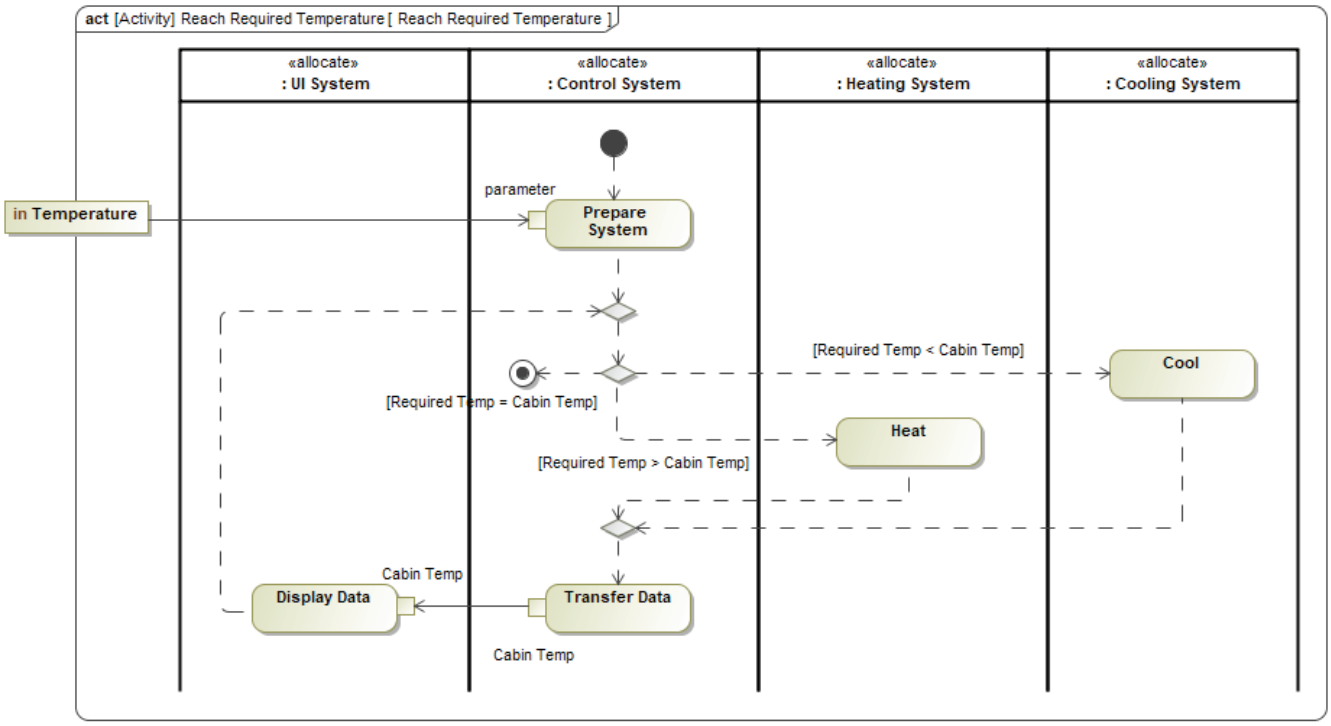
4. Drag the selection onto the empty area of the open diagram pane. Blank swimlane partitions with headers named as corresponding part properties are created and displayed in the diagram (see the following figure).

ⓘ If you want the sequence of partitions to be the same as in the following figure, select the header of the *Control System* partition and click the Move Swimlane Left button ◀ twice. As a result, it becomes the second partition in the succession.



5. Right-click any of the swimlane partitions and select **Allocation Mode** > **Usage**. The allocation to usage mode is enabled in the diagram.
6. Drag the *Prepare System* and *Transfer Data* actions, the initial and final nodes, both merge nodes, and the decision node to the *Control System* partition.
7. Drag the *Heat* action to the *Heating System* partition.
8. Drag the *Cool* action to the Cooling System partition.
9. Drag the *Display Temperature* to the *UI System* partition.

When you're done, your SysML activity diagram *Reach Required Temperature* should look very similar to the one below.



Even though more detailed than the black-box one defined in step 5 of the B2 tutorial, this is still a highlevel scenario and should be decomposed as well. For this, you should decompose every function by performing as many iterations of the W2.initial, W3, and W2.final steps as necessary.

W4. MoEs for Subsystems

What is it?

In this cell, you should specify the MoEs for one or more subsystems of the SoI to make further refinements of non-functional stakeholder needs. This cell is optional, as you might not need to specify MoEs additional to those defined in the B4 cell.

Who is responsible?

In a typical multidisciplinary systems engineering team, MoEs are specified by the *Requirements Team*.

How to model?

To capture MoEs for subsystems in the modeling tool, you can utilize the infrastructure of the SysML bdd – the same way that you used it for capturing MoEs of the SoI in the B4 cell. There is a chance that in this cell you don't even need to create a new bdd; you can exploit the one created for specifying interfaces of the subsystem. For better understanding, remember the example given at the end of step 6 of the W3 tutorial. Imagine that the Cooling System already has interfaces specified in the bdd created directly under the Cooling System package (see step 3 of the W3 tutorial). This bdd can later serve another purpose; i.e., for capturing MoEs of the Cooling System.

What's next?

- Having the measurements of effectiveness, you can specify design constraints, which can be used to verify the system design (see S4).

Tutorial

B1-W1. Stakeholder Needs: final

What is it?

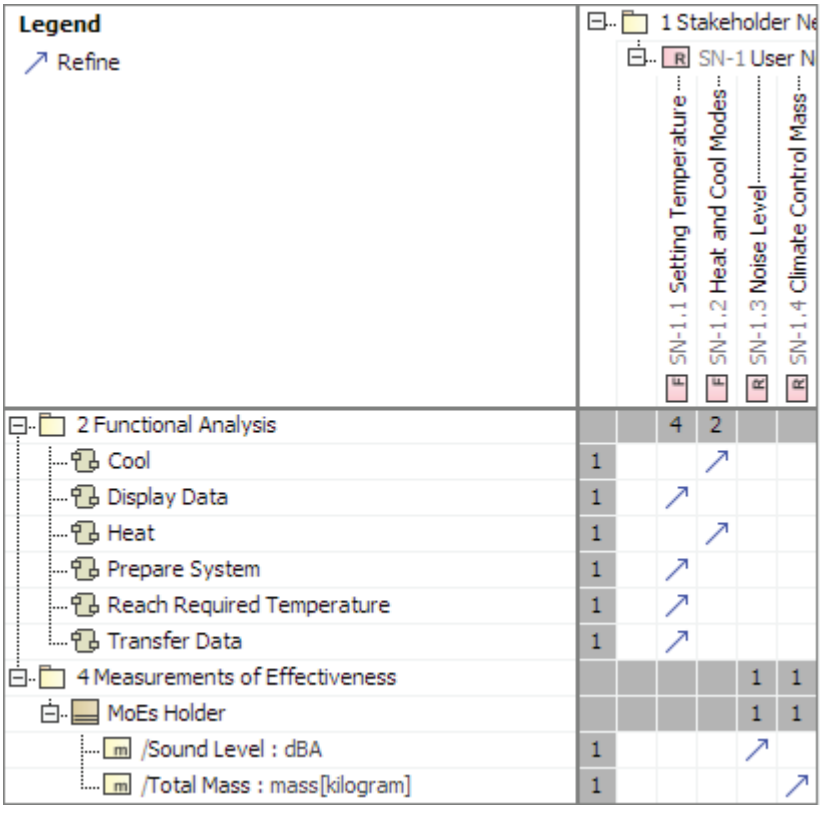
In order to have a complete problem domain model, you need to establish traceability relationships between the elements that capture stakeholder needs and the elements that capture the functions, logical architecture, and non-functional characteristics of the SoI. Since the latter are more concrete than stakeholder needs, we recommend establishing refine relationships. These relationships should point from the elements of the lowest level of detail, for example, activities that capture leaf functions of the functional decomposition model. In an ideal case, every item of the stakeholder needs must be refined by one or more elements of the other SysML types.

Who is responsible?

In a typical multidisciplinary systems engineering team, traceability relationships are established by the *Requirements Team*.

How to model?

Neither of the SysML diagrams is suitable for creating a mass of cross-cutting relationships, such as «refine». For this, a matrix or a map is more suitable. The modeling tool offers a wide variety of predefined matrices for specifying cross-cutting relationships. To capture «refine» relationships, a Refine Requirement Matrix can be used.



What's next?

- You can move to specifying system requirements, which is described in S1.initial.

Tutorial

Step 1. Creating a matrix for capturing refine relationships

To get ready for capturing «refine» relationships, you should create a Refine Requirement Matrix first. Since the matrix is used to specify the refinements of stakeholder needs stored in the model under the *1 Stakeholder Needs* package (as requirements), it can be created right there too.

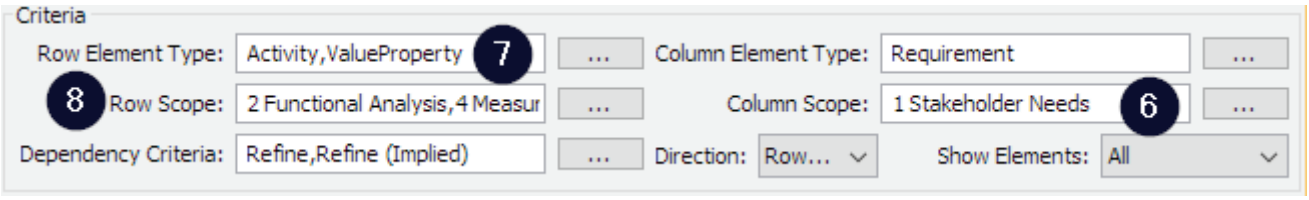
The matrix is predefined to represent requirements in its columns and refine relationships in its cells. You only need to specify that the matrix displays activities and value properties (with «moe» stereotype) in its rows, and that

- Column elements are taken from the *1 Stakeholder Needs* package
- Row elements are taken from the *2 Functional Analysis* package and the *4 Measurements of Effectiveness* package

To create a Refine Requirement Matrix

1. Expand the *1 Problem Domain* and *1 Black Box* packages in succession, if not yet expanded.
2. Right-click the *1 Stakeholder Needs* package and select **Create Diagram**.
3. In the open dialog, click the **Expert** button to turn on the appropriate mode.
4. In the search box, type *rrm*, the acronym of the Refine Requirement Matrix, and then press Enter. The matrix is created.
5. Type *Functions and MoEs to SNs* to specify the name of the new matrix and press Enter again.
6. Specify the column element type:
   a. In the Model Browser, select the *1 Stakeholder Needs* package.
   b. Drag it onto the **Column Scope** box in the **Criteria** area above the matrix contents. The selected package becomes the scope of the matrix columns (see the following figure).
7. Specify row element type:
   a. In the Model Browser, select any activity; for example, the *Cool* activity.
   b. Holding down the Ctrl key, select any value property; for example, the *Total Mass* value property.
   c. Drag the selection onto the **Row Element Type** box in the **Criteria** area above the matrix contents. The rows of the matrix are set to display the elements of the selected types (see the following figure).
8. Specify the row element scope:
   a. In the Model Browser, select the *2 Functional Analysis* package.
   b. Holding down the Ctrl key, select the *4 Measurements of Effectiveness* package.
   c. Drag the selection onto the **Row Scope** box in the **Criteria** area above the matrix contents. The selected packages become the scope of the matrix rows (see the following figure).

The following figure displays the **Criteria** area of the matrix, with highlights on the step 6, 7, and 8 related criteria.

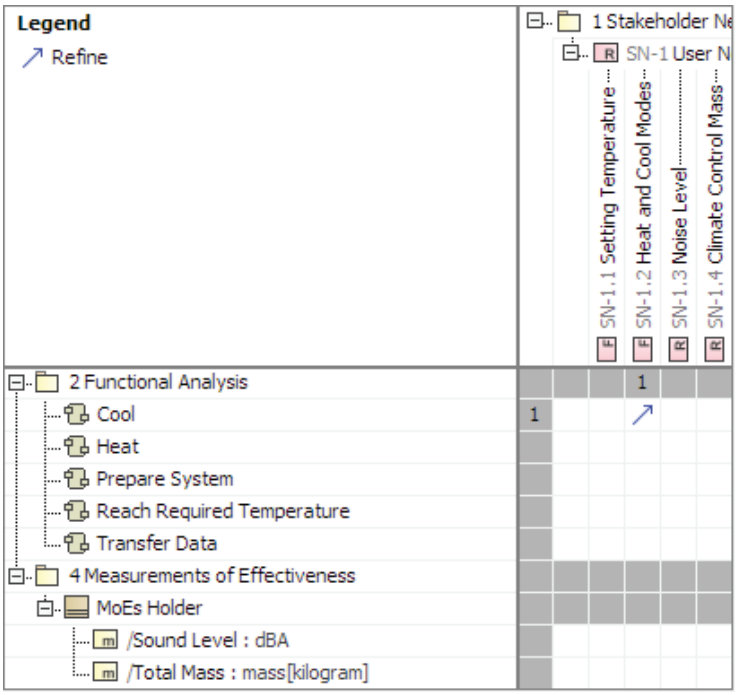


Step 2. Capturing refine relationships

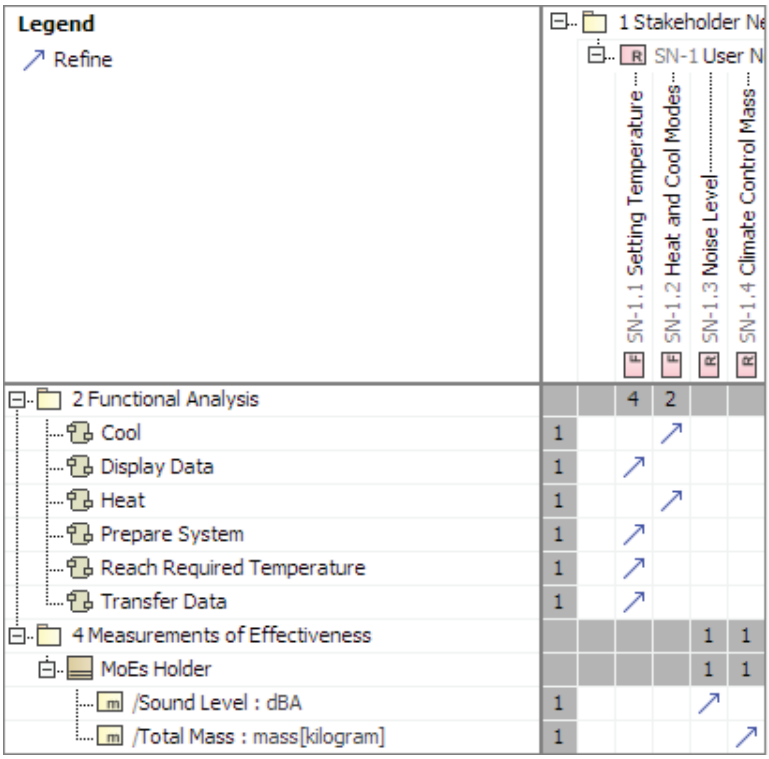
Now you're ready to capture refine relationships. Let's create the one between the *Cool* activity and the *SN-1.2 Heat and Cool Modes* stakeholder need (captured as a requirement) to convey that the former refines the latter.

To specify the refine relationship in the matrix

- Double-click the cell at the intersection of the row that displays the *SN-1.2 Heat and Cool Modes* requirement and the column that displays the *Cool* activity. The derivation relationship is created between the appropriate items and displayed in the cell.



After all refine relationships are captured, your *Functions and MoEs to SNs* matrix should be like the one in the following efined by one or more elements of the other SysML types.

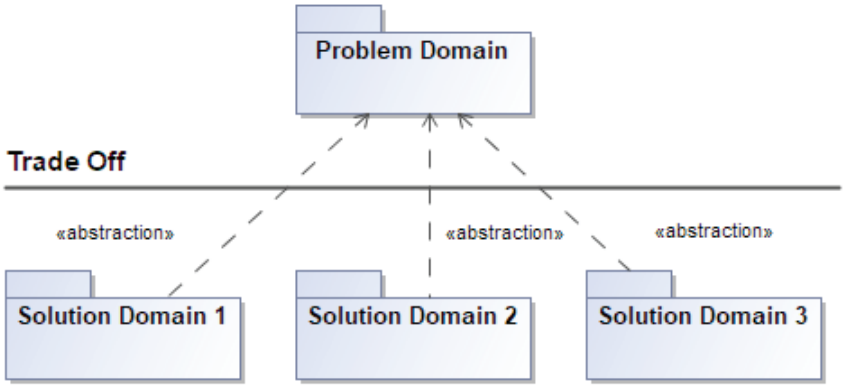


Solution domain

Introduction

Once the problem domain analysis is completed and the stakeholder needs are transferred into the model, it's time to start thinking about the solution. The solution architecture defines a precise model of the logical design of the system or even several variations of it. In other words, this layer of abstraction provides one or more solutions for the problem defined in the first layer of abstraction (including both the black- and white-box perspectives). Having several solutions, a trade-off analysis can be performed to choose the optimal one for implementing the system.

It should be noted that problem domain can be specified by one organization, and solution architecture by another. Furthermore, diverse solutions can be provided by separate organizations.



As you can see in the following table, building the solution architecture consists of specifying requirements, behavior, structure, and parameters of the system under design (no longer system of interest!). Furthermore, the task of building a solution architecture usually consists of more than one iteration, going down from the system-level to the subsystem-level, from the subsystem-level to the component-level architecture and even deeper, if there is a need identified. The precision of the system architecture model depends on the number of iterations.

	PILLAR						
DOMAIN			**Requirements**	**Behavior**	**Structure**	**Parameters**	**Specialty Engineering**
	Problem	**Black Box**	B1-W1 Stakeholder Needs	B2 Use Cases	B3 System Context	B4 Measurements of Effectiveness	
		White Box		W2 Functional Analysis	W3 Logical Subsystems Communication	W4 MoEs for Subsystems	
	Solution		S1 System Requirements	S2 System Behavior	S3 System Structure	S4 System Parameters	**Integrated Testing**
			SS1 Subsystem Requirements	SS2 Subsystem Behavior	SS3 Subsystem Structure	SS4 Subsystem Parameters	
			...	...	...	...	**Analysis**
			C1 Component Requirements	C2 Component Behavior	C3 Component Structure	C4 Component Parameters	
	Implementation		I1 Physical Requirements	Software, Electrical, Mechanical			

Deploying solution domain

Solution architecture can be specified in a single model and even in the same model with the problem domain definition. However, in most cases, the scope and complexity of the solution architecture exceeds the scope and complexity of the problem domain definition up to several times, especially when it includes more than one solution. Defining the solution architecture in two or more separate models helps to manage the complexity. Moreover, such model partitioning enables a more granular model ownership, change analysis and control, access control, concurrent modifications, and model reuse. The following material describes an approach of deploying the solution architecture across multiple models. Though this is not a part of the MagicGrid approach, you're welcome to get familiar with it and adopt it in your organization, if you find it worthwhile.

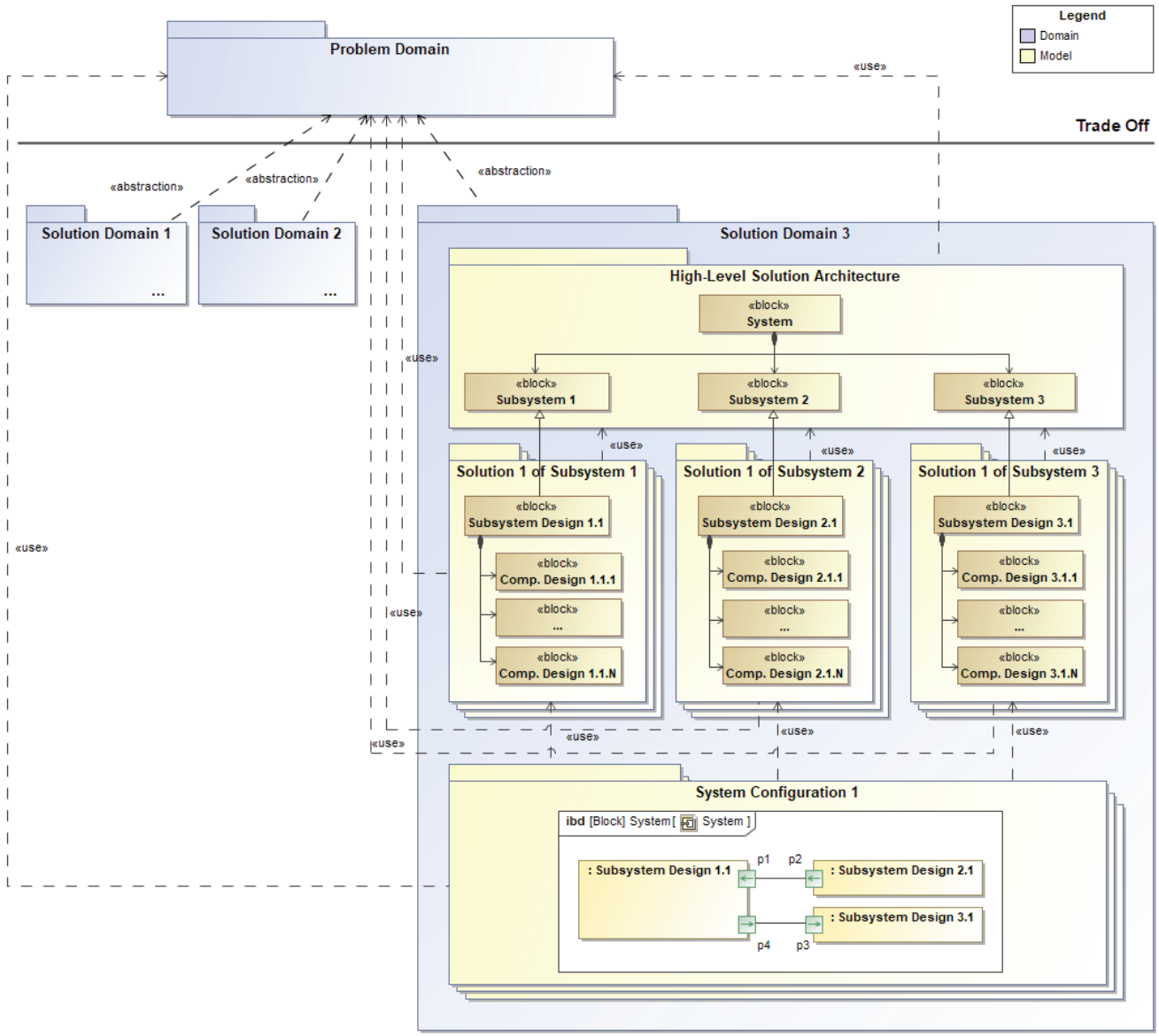
As was stated before, more than one solution architecture can be provided for the same problem. To explain our approach, we introduce the case of three solution architectures: *Solution Domain 1*, *Solution Domain 2*, and *Solution Domain 3*. Though the contents of the *Solution Domain 1* and *Solution Domain 2* are suppressed, you should consider them similar to the contents of the *Solution Domain 3*. The *High-Level Solution Architecture (HLSA)* model is the core of each solution architecture. Based on the results of the problem domain analysis, it defines the subsystems of the system under design in a single-level hierarchy. As you can see in the following figure, there are three subsystems in this case: *Subsystem 1, Subsystem 2,* and *Subsystem 3*. The *HLSA* model also specifies interfaces of each subsystem to determine how they inter-operate with one another and integrate into the whole. Interfaces are not included in the figure below, to avoid clutter. The owner of the *HLSA* model is the systems engineer (also known as the systems architect), the one, who has "a big picture view" of the system under design. In other words, the systems engineer is like the conductor of an orchestra: he/she ensures everyone is playing the same melody at the same time.

Definition of the subsystems in the *HLSA* model helps to identify work packages and allocate them to separate engineering teams. Every team develops one or more solution architectures for the allocated subsystem. It may happen that they work in parallel. As you can see in the following figure, each of the three subsystems has several solutions defined: *Solution* 1 is visible on the top and the others are hidden underneath. The generalization relationship (solid line with hollow, triangular arrowhead) between *Subsystem 1* and *Subsystem Design 1.1* means that the engineering team working on the *Solution 1* of *Subsystem 1* model knows (or speaking in terms of SysML, inherits) the design constraints (such as interfaces) of the subsystem defined in the *HLSA* model, and must follow them. Other generalizations convey appropriate information. Solution architectures of different subsystems and even different solution architectures of the same subsystem may include a diverse number of components. A solution architecture of the subsystem defines the requirements, behavior, structure, and parameters of that subsystem.

After engineering teams are done with the solution architectures of the subsystems, the systems engineer can integrate them into a single model. He/she picks up the preferred one of each subsystem and builds the integrated solution architecture of the whole system. Actually, more than one solution can be proposed. As you can see in the following figure, we call them *System Configuration 1, System Configuration 2,* etc. Just like the solution architecture of every subsystem, the integrated solution architecture of the whole system under design defines the behavior, structure, and parameters, and must meet system requirements specification.

Having the full model of the system under design, the systems engineer can perform the trade-off analysis and pick up the optimal system configuration; having optimal system configurations of several solution architectures, the systems engineer can pick up the optimal solution above all. This is illustrated by the *Trade Off* analysis in the figure below.

It's important to mention that trade offs can be done not only between solution architectures, but at every level of granularity in the solution architecture.



S1. System Requirements: initial

What is it?

To be able to design a system architecture, you first need to have the system requirements specification in order to follow it. Results of the problem domain analysis from both black- and white-box perspectives should serve you as input for producing the system requirements specification. System requirements are derived from stakeholder needs, and they also refine the elements captured in cells B2–B4 and W2–W4.

Modeling in this cell includes two phases: initial and final. During the initial phase, system requirements are captured and the following traceability relationships are established:

- From system requirements to stakeholder needs – to assert what system requirements are derived from what stakeholder needs

- From system requirements to the rest of the problem domain model – to assert what elements are refined by what system requirement

After you're done with the system architecture described in cells S2, S3.initial, S3.final, and S4, you should get back to system requirements and establish satisfy relationships from system design elements to system requirements in order to assert which of the former fulfill which of the latter. This is the final phase of modeling in this cell.

The same rules of modeling and setting traceability relationships apply for subsystem requirements, component requirements, and even those of deeper structure. Just like system requirements are derived from stakeholder needs, subsystem requirements should be derived from system requirements, component requirements from subsystem requirements, and so on. Just like system requirements are satisfied by system design elements, subsystem requirements should be satisfied by subsystem design elements, component requirements by component design elements, and so on.

Who is responsible?

In a typical multidisciplinary systems engineering team, the system requirements are specified by the *Requirements Team*.

How to model?

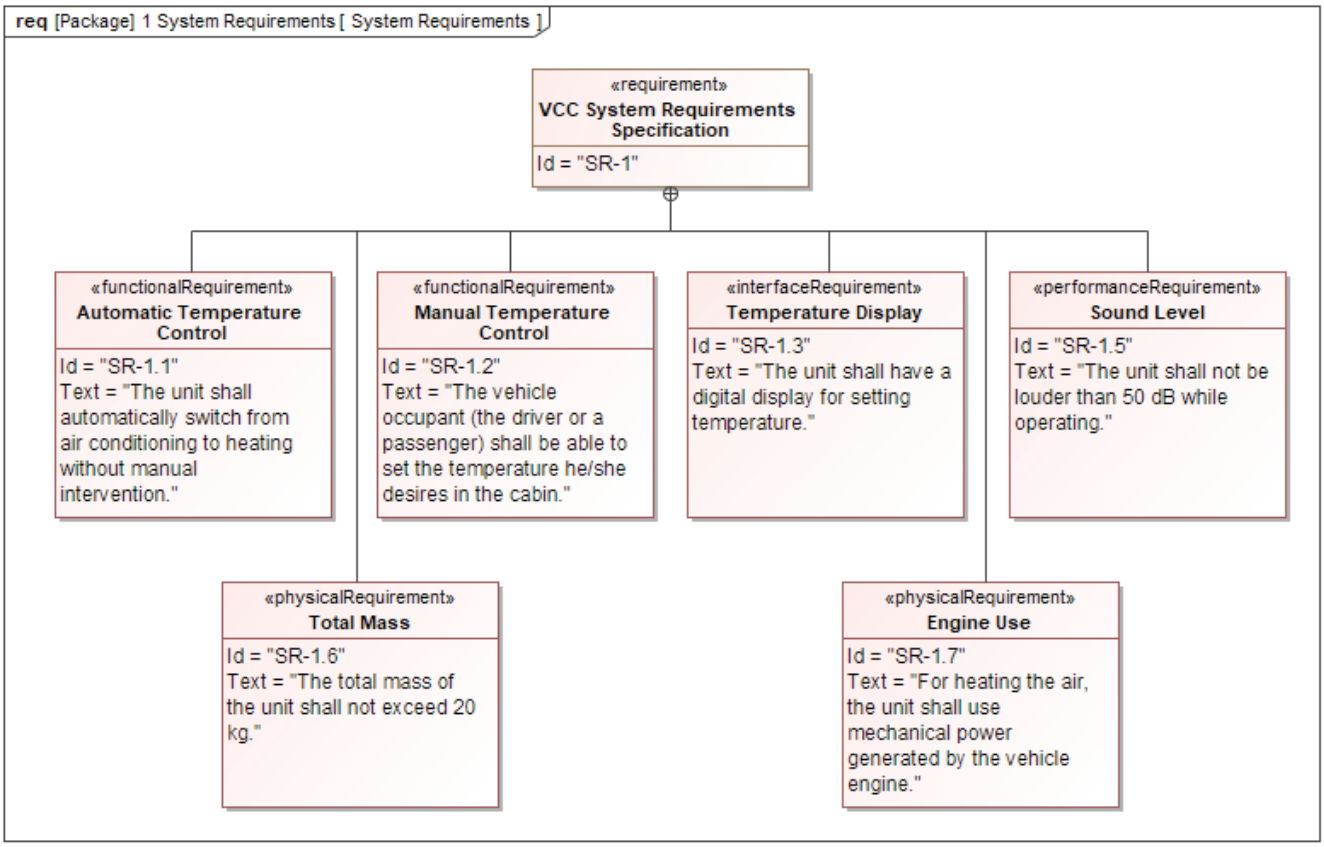
In the modeling tool, an item of system requirements specification can be stored as the SysML requirement, which has a unique identification, name, and textual specification. Each system requirement must cover a certain item of stakeholder needs. For this, you can use the «deriveReqt» relationship pointing from the system requirement to that item of stakeholder needs. Moreover, each system requirement must refine one or more elements from the problem domain model. For this, you can use the «refine» relationship pointing from the system requirement to some block, activity, or MoE, to name a few.

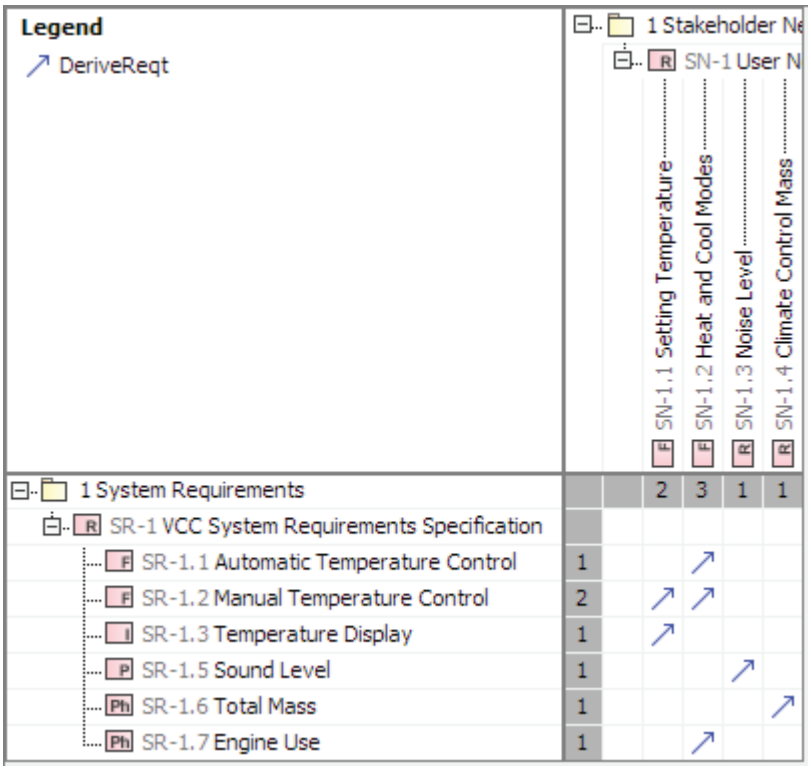
As opposed to the textual specifications of stakeholder needs, the ones of system requirements must be formal and written down by following certain guidelines; for example, write in short and clear sentences only, avoid conditional keywords, such as *if, except, as well*; use the *shall* keyword; and include tables. For more information on how to write good textual requirements, refer to *INCOSE Systems Engineering Handbook*.

System requirements can be organized into one or more hierarchy levels. For this, use the containment relationships among them. The SysML requirement referring to the system requirements specification must be at the top of the hierarchy. It may have no text.

The SysML requirement table or diagram can help to visualize the hierarchy of system requirements. They can also be included in the requirements report. We recommend having a diagram or a table for displaying the hierarchy of system requirements (see the first figure below), and another diagram or a matrix for establishing and visualizing various

traceability relationships; for example, derivation between system requirements and stakeholder needs (see the second figure below).





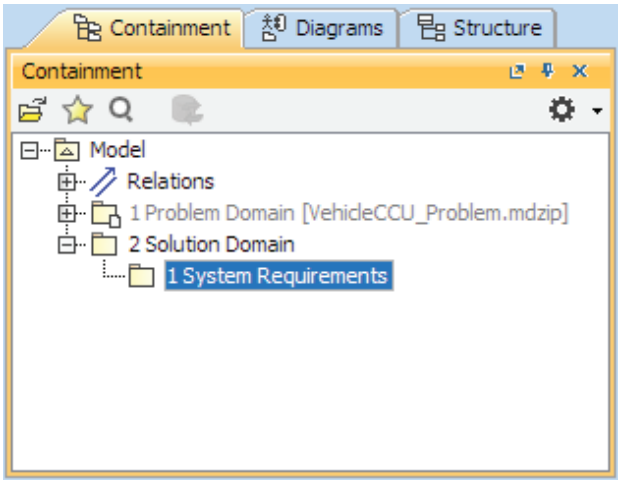
What's next?

When you have the system requirements specification, you can start building the system architecture; that is, move to S3.initial.

Tutorial

Step 1. Creating and organizing a model for S1 initial

To get ready for specifying system requirements, you need to establish an appropriate structure of the model beforehand. Following the design of the MagicGrid framework, model artifacts that capture system requirements should be stored under the structure of packages displayed in the following figure. As you can see, the upper-level package represents the domain, and the lower-level package stands for the cell.



However, in real-world projects it is quite common to keep the solution domain model separated from the problem domain model. Following this practice, we recommend specifying the solution architecture of the Vehicle Climate Control System (VCCS), generally referred to as the system under design, in another model (file). The contents of the problem domain model should be available in that model, because the results of the problem domain analysis should be used as the basis for the system requirements specification. Speaking in terms of the modeling tool, the problem domain model *should be used* in the solution domain model. For this, we utilize the relevant capabilities of the modeling tool.

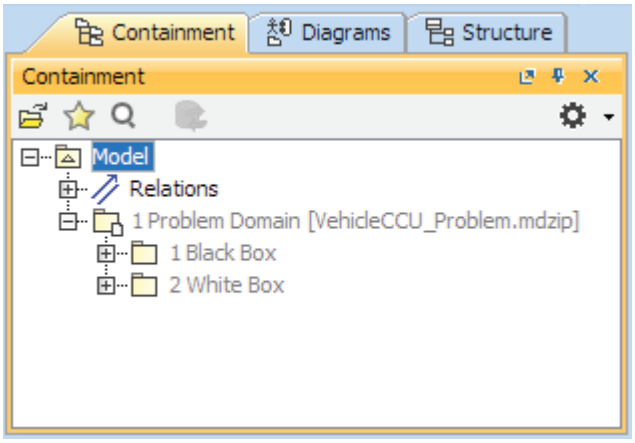
To create and organize the model for S1 initial

1. Start sharing the problem domain model.

   ⓘ For more information on this topic and the topics mentioned in steps 2 and 3, refer to the latest documentation of your modeling tool.

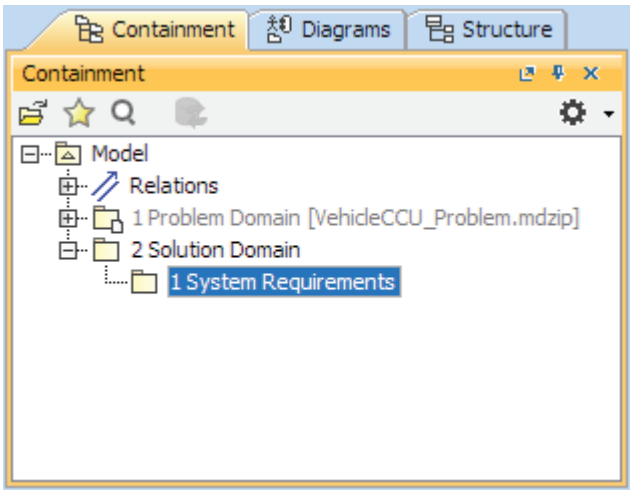
2. Create a new model (file). It can be named *VehicleCCS_Solution.mdzip*.

3. Use the problem domain model as *read-only* in the solution domain model. This is the solution domain model of the system.



> ⓘ Names in grey indicate that elements cannot be modified. This is because the problem domain model is used in the solution domain model as *read-only*.

4. Right-click the *Model* package (this is the default name of the root package) and select **Create Element**.
5. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.
6. Type *2 Solution Domain* to specify the name of the new package and press Enter.
7. Right-click the *2 Solution Domain* package and select **Create Element**.
8. Repeat step 5.
9. Type *1 System Requirements* to specify the name of the new package and press Enter.



Step 2. Creating a diagram for system requirements

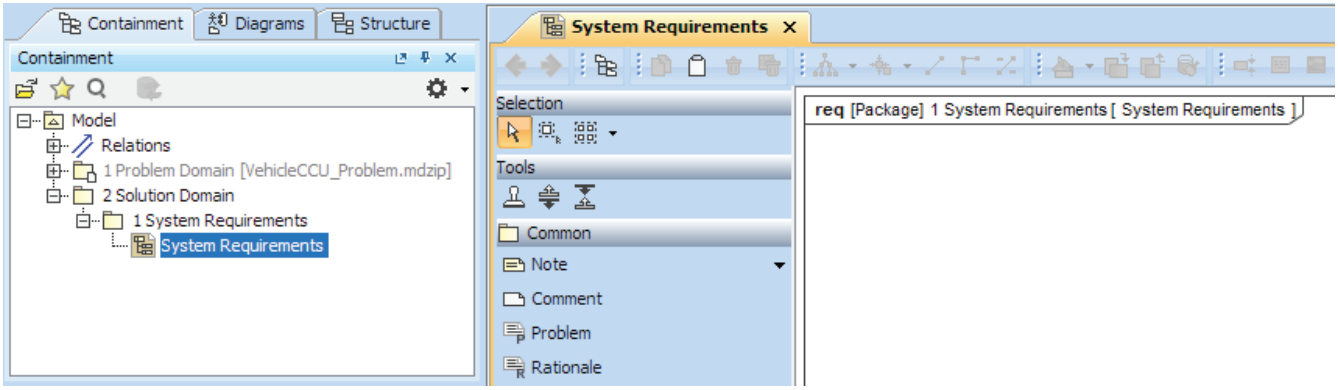
For capturing and displaying system requirements, we recommend using the SysML requirement diagram or table. While using the table is introduced in the tutorial of the B1-W1.initial phase, using the diagram is described as follows.

To create the SysML requirement diagram for specifying system requirements

1. Right-click the 1 *System Requirements* package and select **Create Diagram**.

2. In the search box, type *rd*, where r stands for *requirement* and *d* for *diagram*, and then doublepress Enter. The diagram is created.

> ⓘ Note that the diagram is named after the package where it is stored. This name works for the diagram, too. You may only remove the sequence number from its name.



Step 3. Specifying system requirements

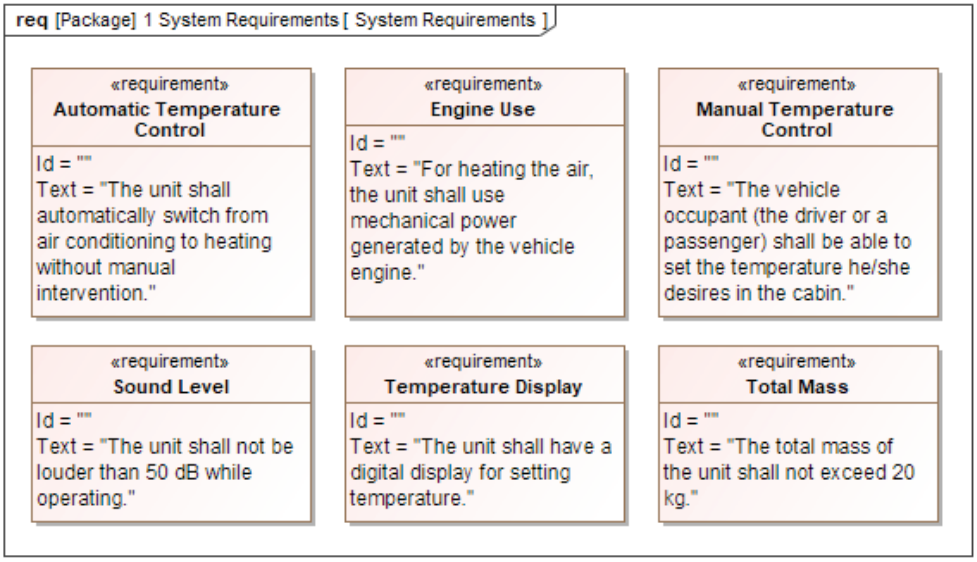
The white-box analysis of the Vehicle Climate Control Unit reveals the following system requirements:

#	Name	Text
1	Manual Temperature Control	The vehicle occupant (the driver or a passenger) shall be able to set the temperature he/she desires in the cabin.
2	Automatic Temperature Control	The unit shall automatically switch from air conditioning to heating without manual intervention.
3	Total Mass	The total mass of the unit shall not exceed 20 kg.
4	Engine Use	For heating the air, the unit shall use mechanical power generated by the vehicle engine.
5	Temperature Display	The unit shall have a digital display for setting temperature.
6	Sound Level	The unit shall not be louder than 50 dB while operating.

To capture the system requirements in your model

1. Open the *System Requirements* diagram, if not opened yet.
2. Click the **Requirement** button on the diagram palette and then click a free space on the diagram pane. An empty requirement is created.
3. Type the name of the first requirement from the table above directly on the shape and press Enter.
4. Click the **Text** property value on the shape once and then again. The property switches to the edit mode.
5. Type the text of the first requirement from the table.
6. Click somewhere outside the shape when you're done.
7. Repeat steps 2 to 6 to capture the rest of the requirements.

After you're done, the contents of the *System Requirements* diagram should look very similar to the one in the following figure.



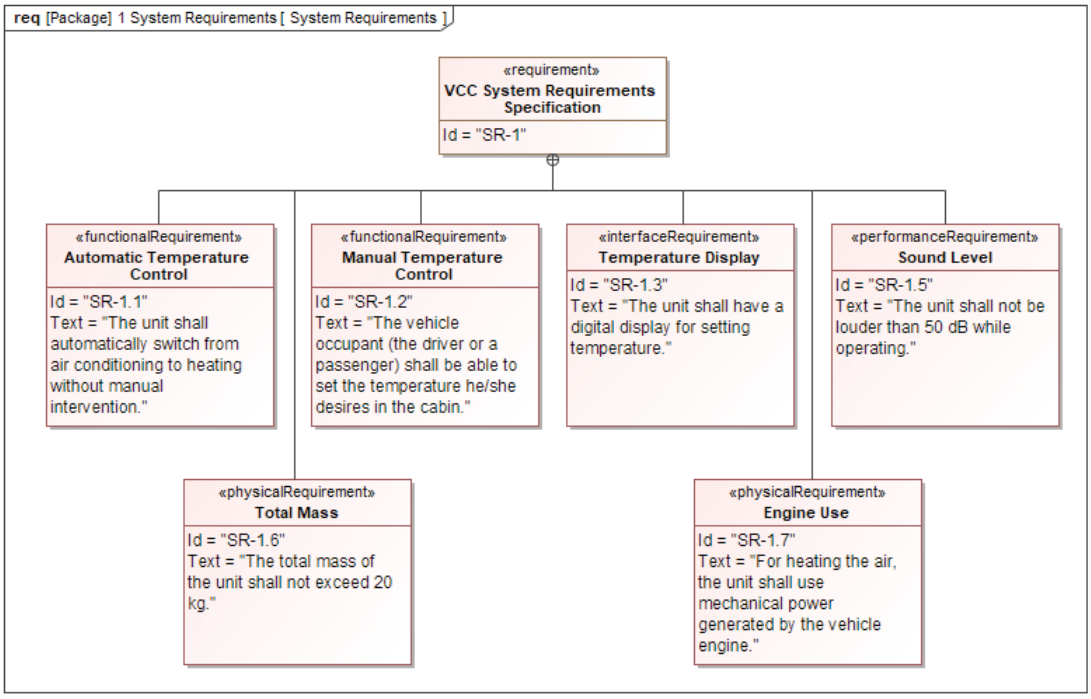
In addition, you need a grouping requirement to represent the entire requirements specification. The easiest way of creating a grouping requirement is described in step 4 of the B1-W1 initial phase tutorial. Follow the procedure to create the grouping requirement *VCC System Requirements Specification*.

To display the requirements hierarchy in the *System Requirements* diagram

1. Drag the requirement *VCC System Requirements Specification* to that diagram pane.
2. Right-click the shape of that requirement and select **Display** > **Display Paths**. Containment relationships among requirements are displayed on the diagram pane.
3. On the diagram toolbar (located on the top of the diagram pane), click the Quick Diagram Layout button  .

In order to distinguish system requirements from stakeholder needs and component requirements, we recommend adding prefixes to their numbers. Follow the procedure described in step 5 of the B1-W1 initial phase tutorial to add the prefix *SR-* to your system requirements. Subsequently, these system requirements can be refactored to more specific ones. For example, the *Manual Temperature Control* requirement can become functional, the *Total Mass* requirement physical, and the *Sound Level* requirement the one of performance. For this, follow the procedure described in step 6

of the B1-W1 initial phase tutorial. After you are done with numbering and categorization, the contents of your *System Requirements* diagram should look very similar to the one in the following figure.



Step 4. Establishing traceability to stakeholder needs

Traceability relationships from system requirements to stakeholder needs can be established either in a SysML requirements diagram or a dependency matrix. We recommend using the dependency matrix for large scope, because it provides a more compact view of the model than the diagram does.

In this case, you need to assert what system requirements (more specific) are derived from what stakeholder needs (more generic), that is, establish derivation or, in terms of SysML, «deriveReqt» relationships between them. Therefore, we recommend utilizing the infrastructure of the Derive Requirement Matrix, one of the predefined dependency matrices in the modeling tool. As opposed to a common dependency matrix, it has certain criteria predefined to speed up building the view.

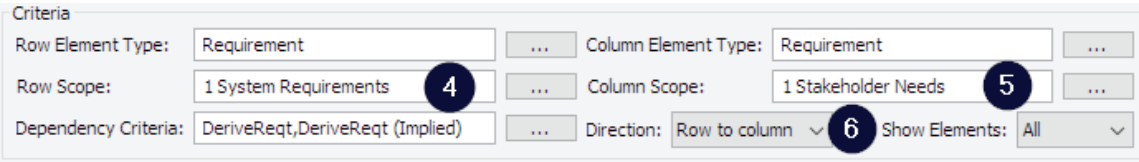
To create a Derive Requirement Matrix

1. In the Model Browser, right-click the 1 *System Requirements* package and select **Create Diagram**.
2. In the search box, type drm, the acronym for the predefined Derive Requirement Matrix, and press Enter.

   ⓘ If you don't see any result, click the Expert button below the search results list. The list of available diagrams expands in the Expert mode.
3. Type *System Requirements to Stakeholder Needs* to specify the name of the new matrix and press Enter again.
4. In the Model Browser, select the *1 System Requirements* package and drag it onto the **Row Scope** box in the **Criteria** area above the matrix contents. The *1 System Requirements* package becomes the scope of the matrix rows.
5. In the Model Browser, select the *1 Stakeholder Needs* package (for this you might need to subsequently expand the *1 Problem Domain* and *1 Black Box* packages first) and drag it onto the **Column Scope** box in the **Criteria** area above the matrix contents. The *1 Stakeholder Needs* package becomes the scope of the matrix columns.
6. Click the **Direction** box and select **Row to Column**, as you need to establish derivation relationships pointing from system requirements, which are row elements, to stakeholder needs, which are column elements.

7. Click the **Refresh** hyperlink in the notification box below the **Criteria** area. The contents of the matrix are updated. All the cells are empty at the moment.

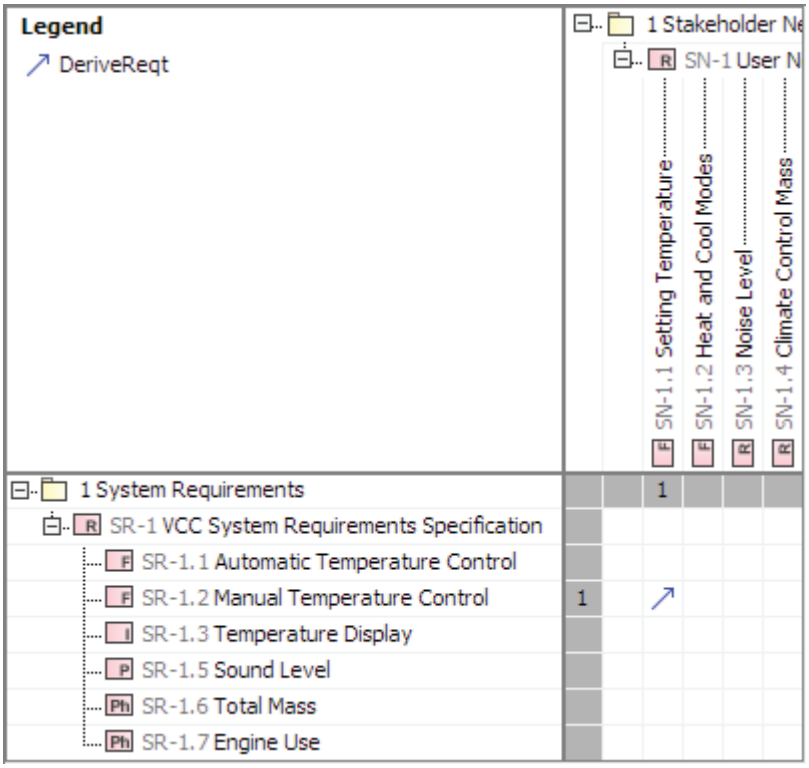
The following figure displays the **Criteria** area of the matrix, with highlights on step 4, 5, and 6 related criteria.



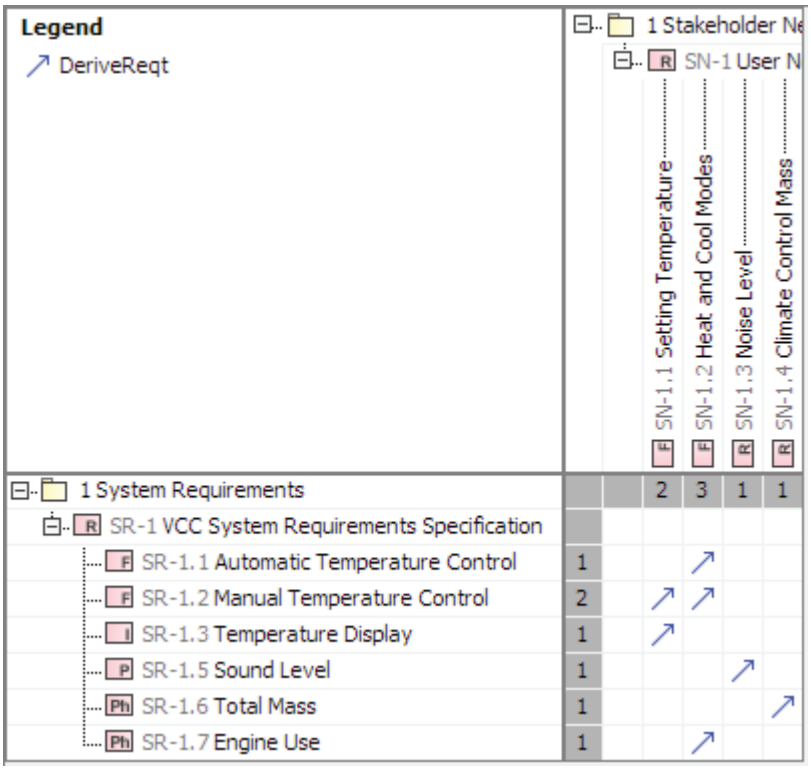
Now you're ready to establish derivation relationships. Let's create the one between *SR-1.1 Automatic Temperature Control* and *SN-1.1 Setting Temperature* to convey that the former is derived from the latter or, in other words, that this system requirement is created because of that stakeholder need.

To specify the derivation relationship in the matrix

- Double-click the cell at the intersection of the row that displays *SR-1.2 Manual Temperature Control* and the column that displays *SN-1.1 Setting Temperature*. The derivation relationship is created between the appropriate items and displayed in the cell.



After all derivation relationships are established, your *System Requirements to Stakeholder Needs* matrix should resemble the one in the following figure. In the ideal case, every system requirement must be derived from one or more stakeholder needs, and every stakeholder need must be covered by one or more system requirements. Otherwise, the system requirements specification must be revised and updated.



Step 5. Establishing traceability to the rest of the problem domain model

Traceability relationships from system requirements to the rest of the problem domain model can be established either in a SysML requirements diagram or in a dependency matrix. We recommend using the dependency matrix for large scope, because it provides a more compact view of the model than the diagram does.

In this case, you need to assert what problem domain elements are refined by what system requirements; that is, establish refine or, in terms of SysML, «refine» relationships between them. Therefore, we recommend utilizing the infrastructure of the Refine Requirement Matrix, one of the predefined dependency matrices in the modeling tool. As opposed to a common dependency matrix, it has certain criteria predefined to speed up building the view.

Let's assume that you want to display blocks capturing system structure and interfaces, activities capturing system functions, and MoEs capturing calculable characteristics of the system as columns, while system requirements are displayed as rows of the matrix.

To create a Refine Requirement Matrix

1. In the Model Browser, right-click the 1 *System Requirements* package and select **Create Diagram**.
2. In the search box, type rrm, the acronym for the predefined *Refine Requirement Matrix*, and press Enter.

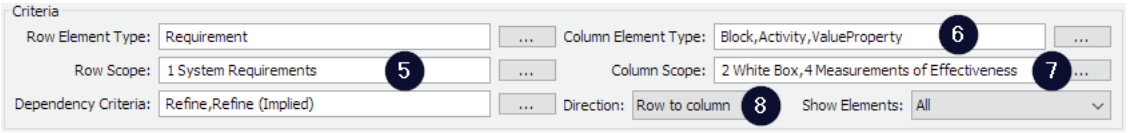
   ⓘ If you don't see any result, click the **Expert** button below the search results list. The list of available diagrams expands in the Expert mode.
3. Type *System Requirements to Problem Domain* to specify the name of the new matrix and press Enter again.
4. On the matrix toolbar, click the **Change Axes** button to swap rows of the matrix with columns. Requirement becomes the type of the matrix rows.

5. In the Model Browser, select the *1 System Requirements* package and drag it onto the **Row Scope** box in the **Criteria** area above the matrix contents. The *1 System Requirements* package becomes the scope of the matrix rows.
6. In the Model Browser, select any block, activity, and value property with the «moe» stereotype and drag the selection onto the **Column Element Type** box in the **Criteria** area above the matrix contents. Blocks, activities, and value properties with the «moe» stereotype become types of the matrix columns.

   ⓘ To select the set of non-adjacent items in the tree, click the first one, press Ctrl, and while holding it down, select other items one by one. This also applies to step 7.

7. In the Model Browser, select the *4 Measurements of Effectiveness* package (for this you might need to expand the *1 Problem Domain* and *1 Black Box* packages first) and the *2 White Box* package, and then drag the selection onto the **Column Scope** box in the **Criteria** area above the matrix contents. Both packages become the scope of the matrix columns.
8. Click the **Direction** box and select **Row to Column**, as you need to establish refine relationships pointing from system requirements, that is row elements, to the artifacts that capture problem domain concepts, which are displayed as column elements.
9. Click the **Refresh** hyperlink in the notification box below the **Criteria** area. The contents of the matrix is updated. All the cells are empty at the moment.

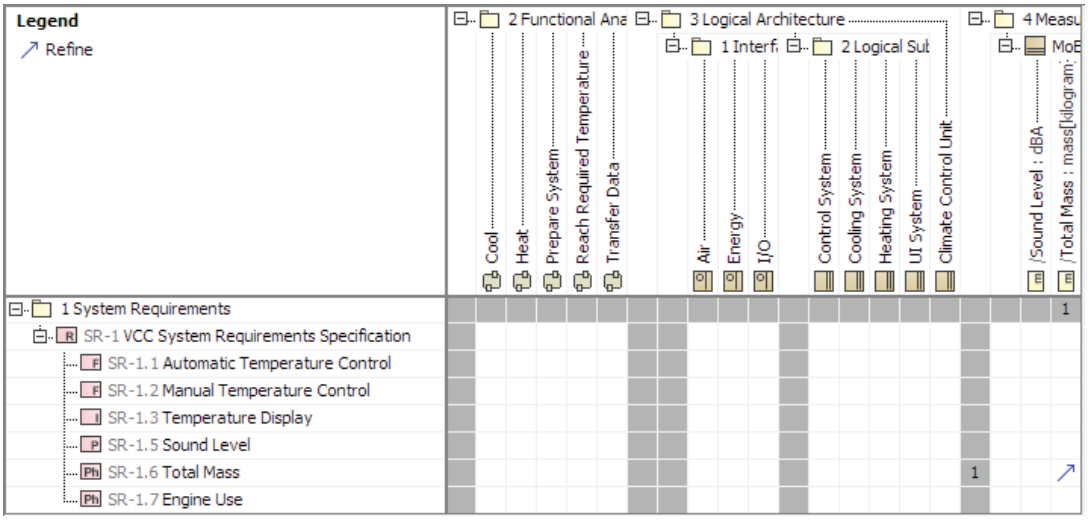
The following figure displays the **Criteria** area of the matrix with highlights on step 5, 6, 7, and 8 related criteria.



Now you're ready to establish refine relationships. Let's create the one between the SR-1.6 Total Mass physical system requirement and the Total Mass MoE to assert that the former refines the latter or, in other words, that this system requirement is created to refine that MoE.

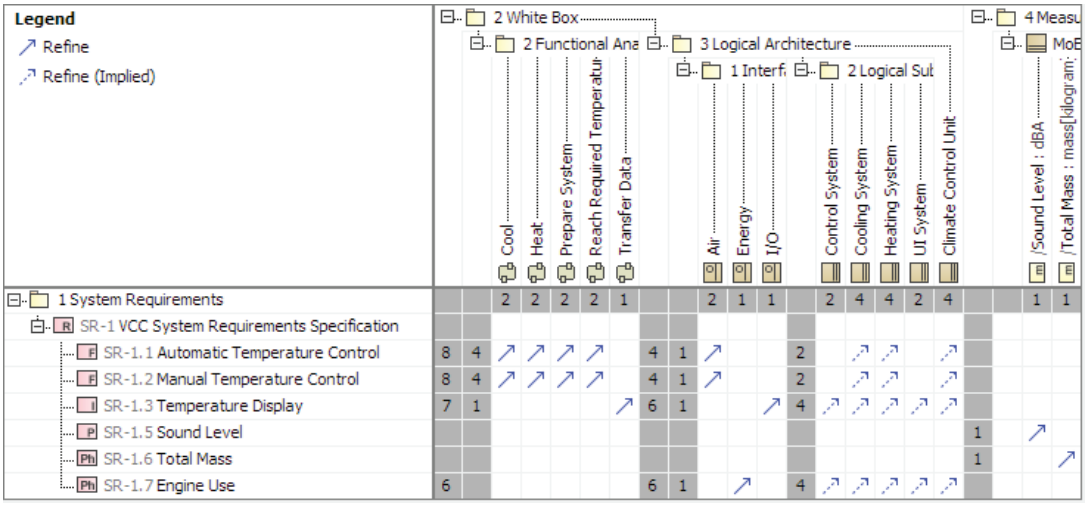
To specify the refine relationship in the matrix

- Double-click the cell at the intersection of the row that displays *SR-1.6 Total Mass* and the column that displays *Total Mass*. The refine relationship is created between appropriate items and displayed in the cell.



After all refine relationships are established, your *System Requirements to Problem Domain* matrix should resemble the one in the following figure. In the ideal case, every system requirement must refine one or more elements captured in the

problem domain model, and every element must be refined by one or more system requirements. Otherwise, the system requirements specification must be revised and updated.



S3. System Structure: initial

What is it?

After the system requirements specification is completed, you can start building the solution architecture of the system under design. Usually it starts with capturing the *HLSA* of the system. The *HLSA* model specifies all the subsystems of the system under design in a single-level hierarchy. The *HLSA* model also specifies interfaces, to determine how these subsystems inter-operate with one another and integrate into the whole. Subsystems of the solution domain may differ from the subsystems identified in the problem domain. This happens quite often, as the problem domain definition is not as precise as the solution architecture. For the purpose of tracing from the subsystems of the solution architecture to the subsystems of the logical architecture in the problem domain model, cross-cutting relationships can be established. Here the first phase of modeling in the S3 cell ends.

Defining the subsystems in the *HLSA* model helps to identify work packages and allocate them to separate engineering teams. Every team develops one or more solution architectures for the allocated subsystem. It may happen that they work in parallel. Modeling of the solution architecture of a single subsystem is described in the SS3 and SS2 cells.

After the engineering teams are done with the solution architectures of subsystems, the systems engineer can integrate them into a single model. As a matter of fact, more than one solution can be proposed; therefore, the systems engineer performs a trade-off study to pick a preferred one of each subsystem and build the integrated solution architecture of the whole system. This is the second phase of modeling in the S3 cell.

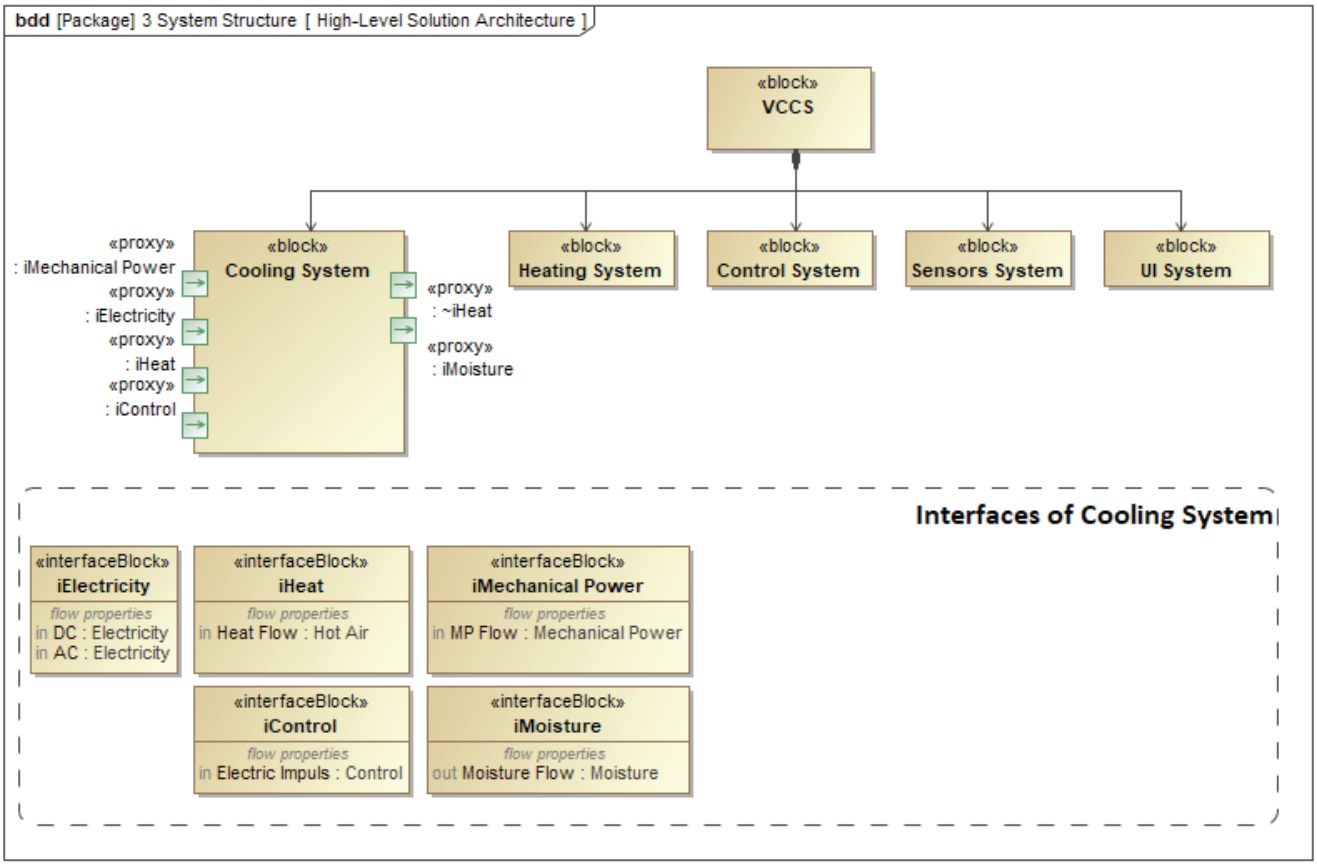
Who is responsible?

In a typical multidisciplinary systems engineering team, the owner of the *HLSA* model is the systems engineer (also known as the systems architect), the head of the *Architecture Team*. Systems engineer is the one, who has “a big picture view” of the system under design and is responsible for the smooth integration of the subsystems into the whole.

How to model?

As was already mentioned, the *HLSA* is a single-level hierarchy that captures the system under design, its subsystems at one level down, and the interfaces to determine how these subsystems inter-operate. The *HLSA* model can be created and displayed in SysML block definition diagram . Subsystems can be captured as blocks, and interfaces as proxy ports typed appropriate interface blocks.

In the following figure you can see a sample *HLSA* model of the Vehicle Climate Control System (VCCS) displaying the blocks of subsystems: *Cooling, Heating, Control, Sensors*, and *UI*. It also displays the interfaces of the Cooling System. Though interfaces of other subsystems are not displayed, we assume they are defined as well.



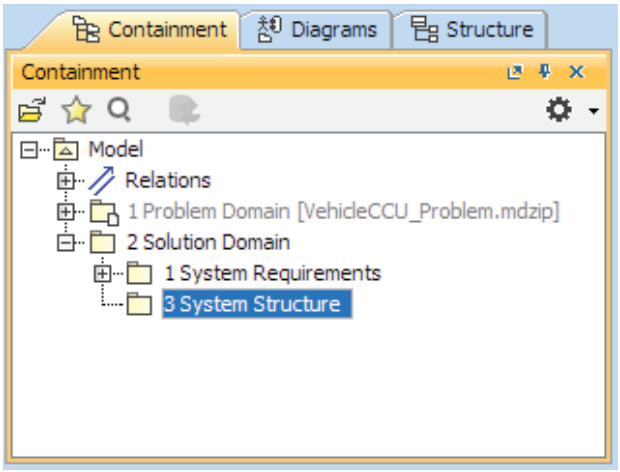
What's next?

- Having logical subsystems, you can start working on their solution architecture models; therefore, it's time to move to the SS3 cell.

Tutorial

Step 1. Organizing the model for S3 initial

The *HLSA* model can be captured in the solution domain model you created in step 1 of the S1 initial phase tutorial. To get ready for building the *HLSA* model, you need to establish an appropriate structure of the model beforehand. Following the design of the MagicGrid framework, model artifacts that capture the *HLSA* of the system should be stored under the structure of packages displayed in the following figure.



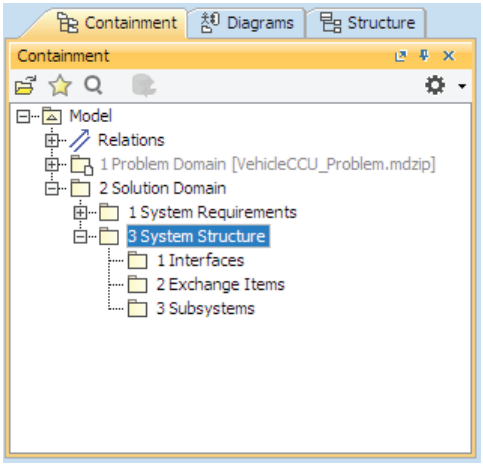
To organize the model for S3

1. Open the solution domain model (the *VehicleCCS_Solution.mdzip* file) you created in step 1 of the S1 initial phase tutorial, if not opened yet.
2. Right-click the *2 Solution Domain* package and select **Create Element**.
3. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.
4. Type *3 System Structure* to specify the name of the new package and press Enter.

For the purpose of keeping the inner structure of the *3 System Structure* package well-organized, you need to create a few more packages. These are:

- *1 Interfaces*
- *2 Exchange Items*
- *3 Subsystems*

When you create them on your own (by following the previous procedure), the Model Browser of your solution domain model should look the same as displayed in the following figure.



Step 2. Creating a bdd for capturing HLSA

To get ready for capturing the High-Level Solution Architecture of the VCCS, you need to create a bdd first.

To create a bdd for capturing the HLSA of the VCCS

1. Right-click the 3 *System Structure* package and select **Create Diagram**.
2. In the search box, type *bdd*, the acronym for SysML block definition diagram, and then press Enter. The diagram is created.
3. Type *High-Level Solution Architecture* to specify the name of the new diagram and press Enter again.

Step 3. Capturing subsystems at HLSA

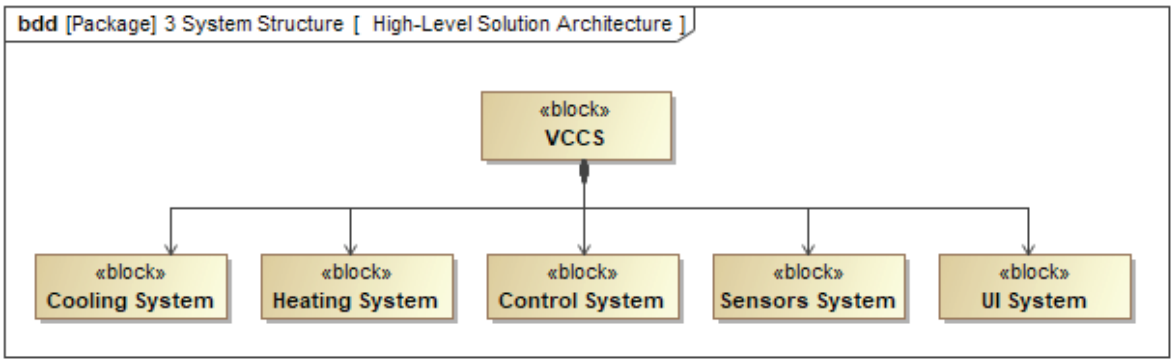
Let's say the system requirements specification reveals that the VCCS consists of:

- Cooling System
- Heating System
- Control System
- Sensors System
- UI System

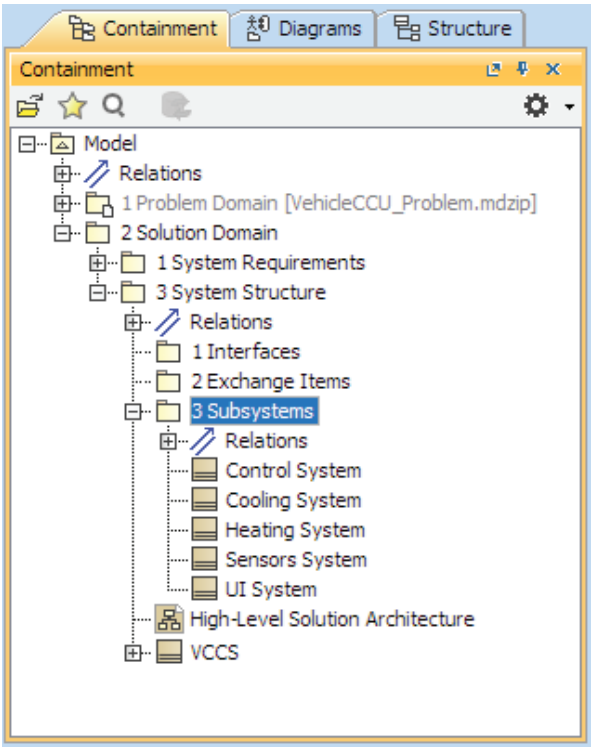
Subsystems can be captured as blocks in the bdd you created in the previous step. As was already mentioned, the *HLSA* is a single-level hierarchy. The detailed design of each subsystem should be subsequently performed by separate engineering teams in isolated models (files).

To capture subsystems of the system under design at HLSA

1. Open the *High-Level Solution Architecture* diagram, if not opened yet.
2. Capture the system under design:
   a. Click the **Block** button on the diagram palette and then click on the diagram pane.
   b. Type *VCCS* to name the block and press Enter.
3. Capture the subsystems:
   a. Click the Direct Composition button on the smart manipulator toolbar of the *VCCS* block and then click an empty place on the diagram pane.
   b. Type *Cooling System* to name the block and press Enter.
   c. Repeat steps 3.a and 3.b to capture other subsystems from the list above.
4. On the diagram toolbar, click the Quick Diagram Layout button .



5. In the Model Browser, select all the blocks of subsystems and drag them to the *3 Subsystems* package you created in step 1 of the S3 initial phase tutorial. The blocks are moved to the *3 Subsystems* package.



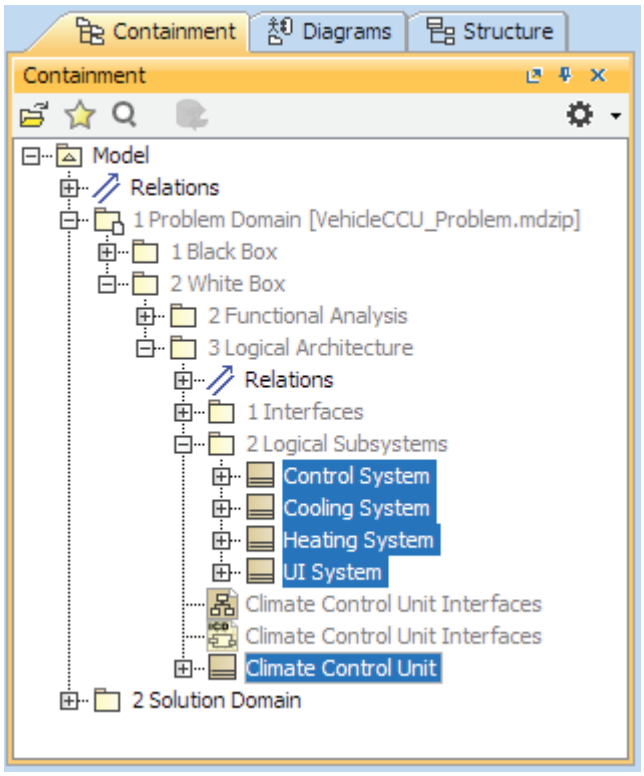
Step 4. Establishing traceability to W3

Traceability relationships enable you to specify what subsystems from the logical architecture (problem domain) determine the creation of what subsystems at *HLSA*. We use the «abstraction» relationships to trace from the solution domain to the problem domain in this case.

To get ready for establishing these relationships, first of all you need to create a separate bdd and display structure blocks from both domains on its pane. The diagram should be stored under the *3 System Structure* package and named, for example, *HLSA to Logical Architecture*. Let's assume that you've already created that bdd on your own. Before establishing the cross-cutting relationships, you should display the blocks you need to link on the diagram pane.

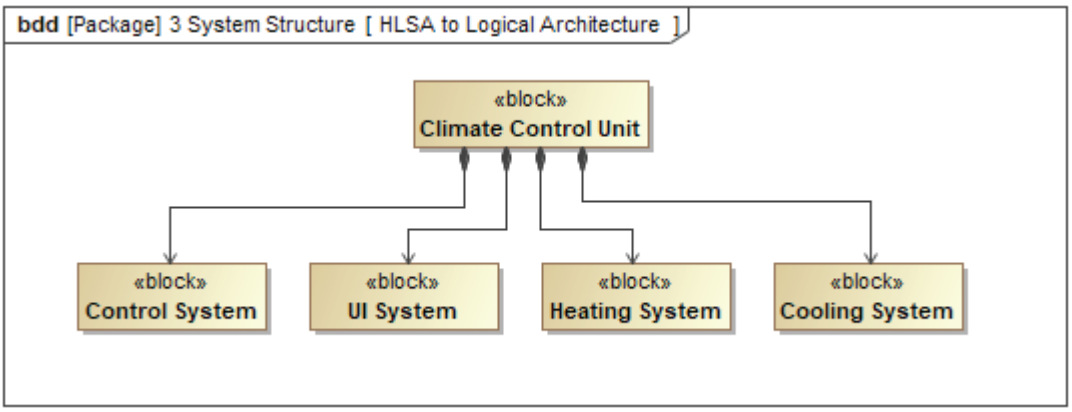
To display the blocks on the *HLSA to Logical Architecture* diagram

1. In the Model Browser, expand the contents of packages in the following sequence: *1 Problem Domain* > *2 White Box* > *3 Logical Architecture* > *2 Logical Subsystems*.
2. Select the *Climate Control Unit* block within the *3 Logical Architecture* package and all the blocks inside the *2 Logical Subsystems* package:
   a. Click the *Control System* block (the first item within the *2 Logical Subsystems* package).
   b. Press Shift and click the *UI System* block (the last item within the *2 Logical Subsystems* package).
   c. Press Ctrl and click the *Climate Control Unit* block.



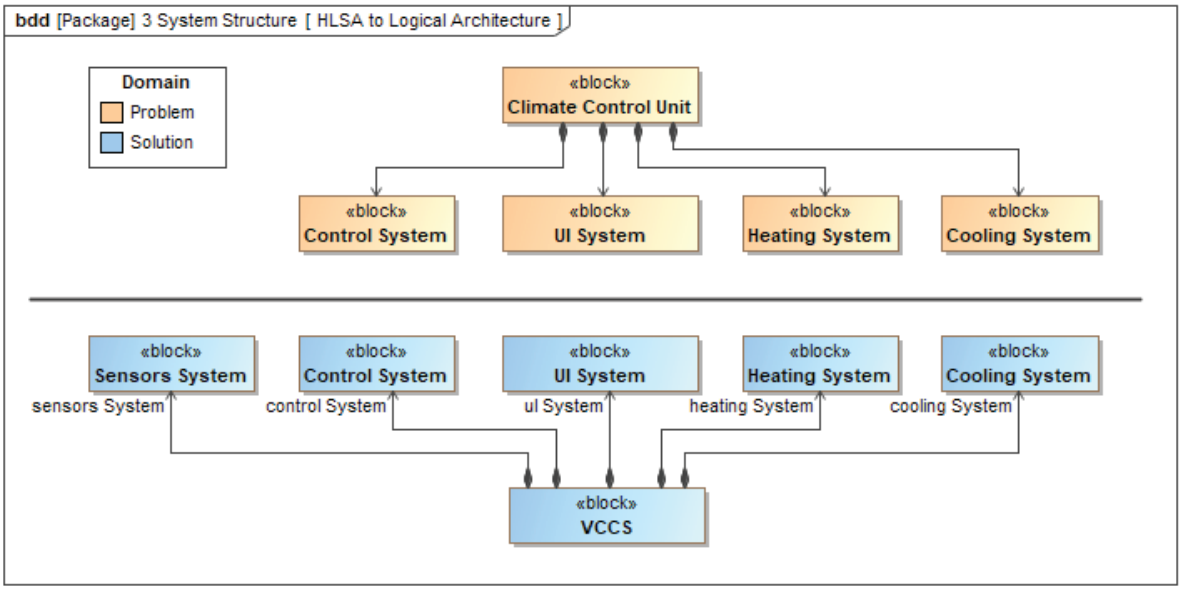
3. Drag the selection to the diagram pane. The shapes of the selected blocks are displayed on the diagram.
4. Right-click these shapes and select **Display** > **Display Paths**.
5. Confirm that you want to display paths between selected symbols only. The paths of direct composition relationships are displayed between the shapes.
6. Click an empty space on the diagram pane to unselect the shapes.

7. On the diagram toolbar, click the Quick Diagram Layout button .



8. In the Model Browser, select all the blocks within the *3 Subsystems* package (you may need to expand the contents of the following packages: *2 Solution Domain* > *3 System Structure* > *3 Subsystems)*.
9. Perform steps 3 to 6 to display the shapes of the selected blocks and paths among them on the diagram.
10. Arrange the shapes so that the layout of the diagram is similar to the one in the following figure.

> ⓘ You can do this manually or by using the adjustment commands. These commands are available after you click the arrow near the Adjust button on the diagram toolbar.



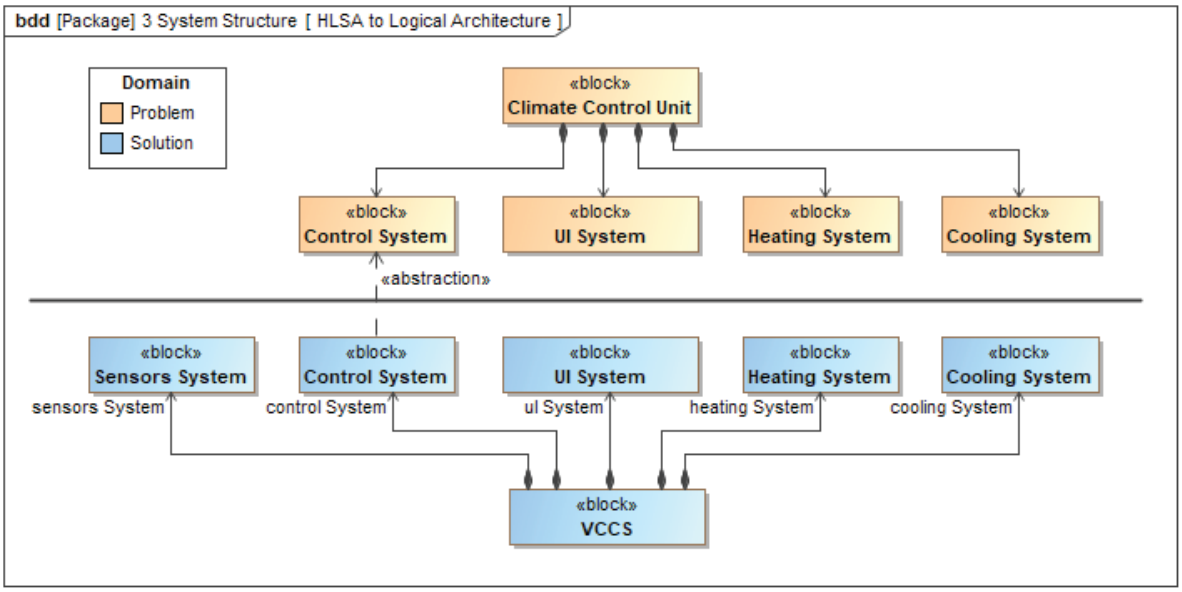
> ⓘ You can make your diagram more informative by adding a
> - Diagram legend, which helps to visually distinguish concepts from different domains.
> - Horizontal separator, which helps to separate these domains.
>
> For more information about how to use these features, see the latest documentation of the modeling tool.

Now you're ready to establish the abstraction relationships. Let's start by creating one between the two blocks capturing the Control System.

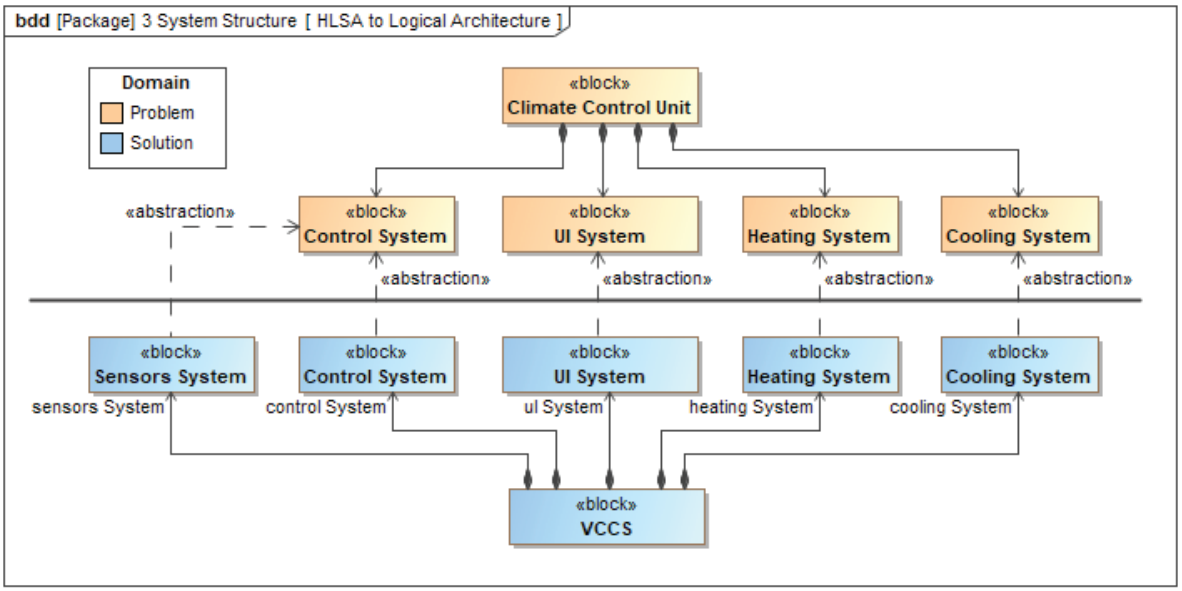
To specify the abstraction relationship

1. Click the **Abstraction** button on the diagram palette.

2. Select the shape of the *Cooling System* block from the solution domain (the bluish one) and then click the one of the *Cooling System* block from the problem domain (the light orange one). The «abstraction» relationship is established between these two blocks.



After all abstraction relationships are created, your *HLSA to Logical Architecture* diagram should be very much like the one in the following figure. Note that the *Control System* has been separated into two in the solution domain: the Sensors System and the *Control System*. That's why blocks capturing these subsystems are related to the same subsystem from the problem domain model.



Step 5. Capturing interfaces at HLSA

Let's say the system requirements specification determines the following interfaces of the Cooling System:

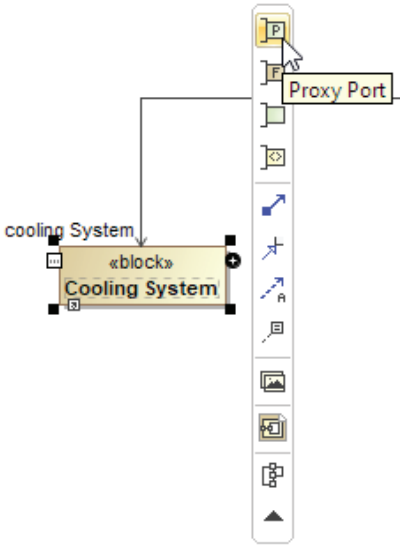
- iElectricity
- iMechanical Power
- iControl
- iHeat
- iMoisture

These interfaces are derived from the logical interfaces of the appropriate functional subsystem (see W3). Interfaces at *HLSA* are more precise, compared to the logical interfaces determined in step 3 of the W3 tutorial. Interfaces are captured as interface blocks that type proxy ports on blocks which capture subsystems. The i prefix in the name of the interface block, which captures a single interface, enables you to easily distinguish interface blocks, which capture interfaces, from other types of blocks in the model.

To determine whether the interface accepts inputs to the subsystem or produces outputs from it, you should specify at least one flow property for that interface.

To capture the interfaces of the Cooling System

1. Make sure the Type Selection Mode is enabled in the *High-Level Solution Architecture* diagram. Otherwise, creation of a proxy port doesn't trigger creation of the interface block to type that proxy port.
2. Select the shape of the *Cooling System* block and on its smart manipulator, click the Proxy Port (see the following figure). A new proxy port appears on the border of the block shape.



> ⓘ Proxy ports are by default named *p1, p2, ..., pn*. It's not necessary to rename them, since they don't convey any semantics in this level of abstraction.

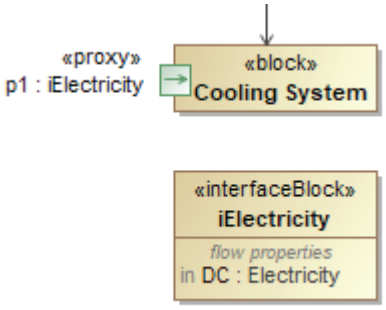
3. Type *iElectricity* next to the colon ("**:**") beside the shape of the proxy port and press Enter.

> ⓘ You can do this only if the Type Selection Mode is enabled in the diagram (see step 1).

A new interface block to type the proxy port is created and displayed in the Model Browser, within the *3 Subsystems* package, where the *Cooling System* block is stored.

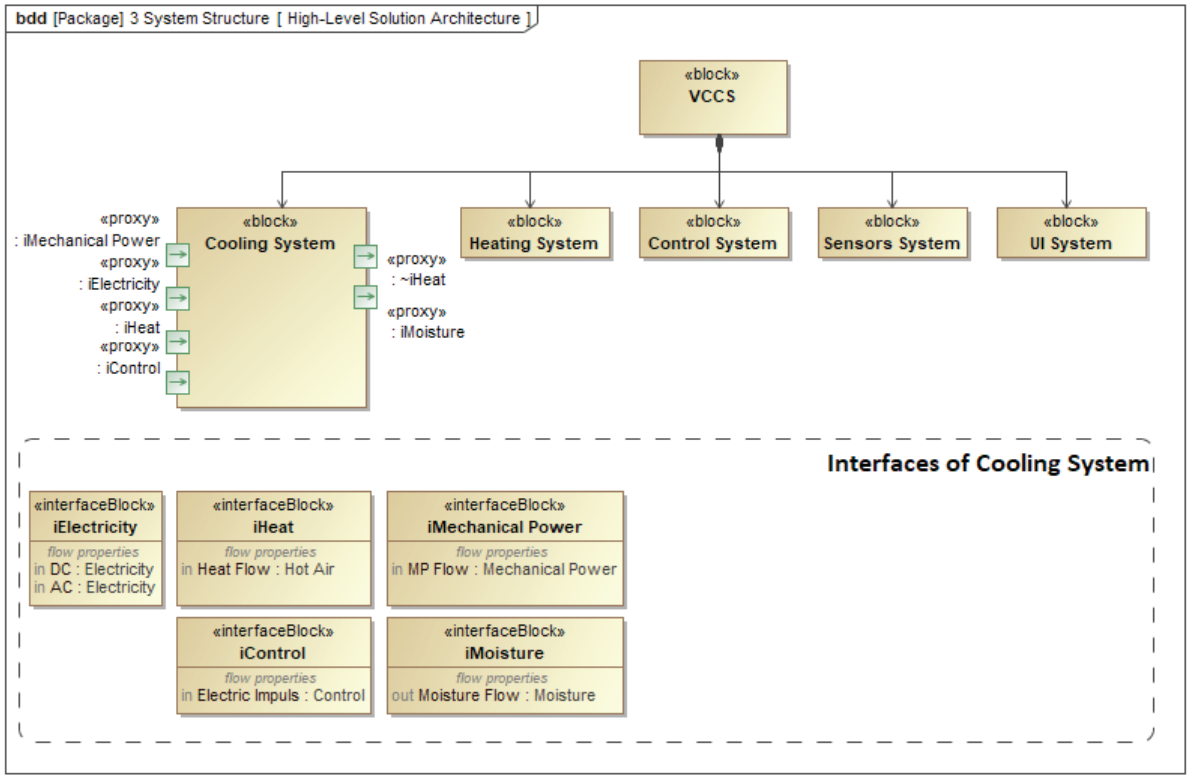
4. Select the *iElectricity* interface block in the Model Browser and drag it onto the diagram pane. A shape of the interface block appears on the diagram.
5. Click the Create Element button on that shape and select **Flow Property**.
6. Directly on the shape of the interface block, do the following:
   a. Remove the *out* part of the flow property direction indicator to specify that the direction of the flow property is *in*.
   b. Type *DC* (*D* for *Direct* and *C* for *Current*) to specify the name of that flow property.
   c. Type *:Electricity* to specify the type of flow property and press Enter.

As a result, an arrow pointing to the inside of the *Cooling System* block appears on the shape of the proxy port (see the following figure). This indicates that the proxy port is typed by the interface block that takes inputs only.

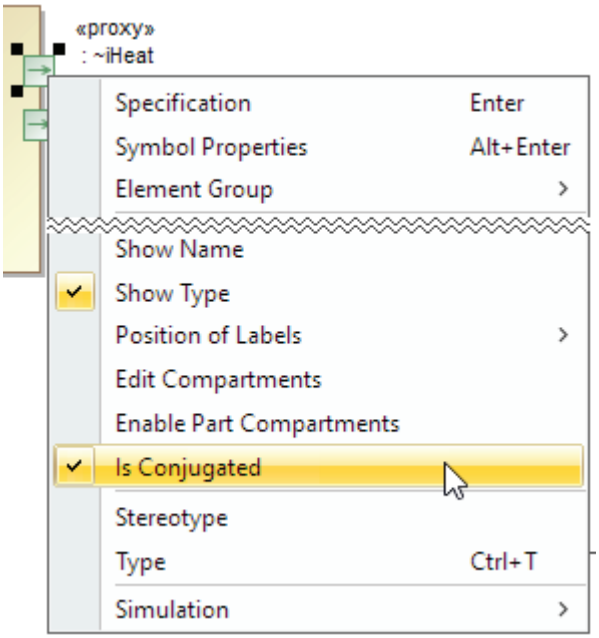


7. Repeat steps 2 to 6 to specify other interfaces. Learn the interface blocks and flow properties you need to create from the following figure.

> ⓘ As you can see, port names are hidden on the diagram. As these names convey no important information, hiding them is a good practice to enhance the diagram's readability. To hide the name of the port, right-click its shape on the diagram pane and unselect the **Show Name** check box. If you want to set this option for all ports at once, select the shape of any port while holding down the Alt key and proceed with the same actions as for a single port.

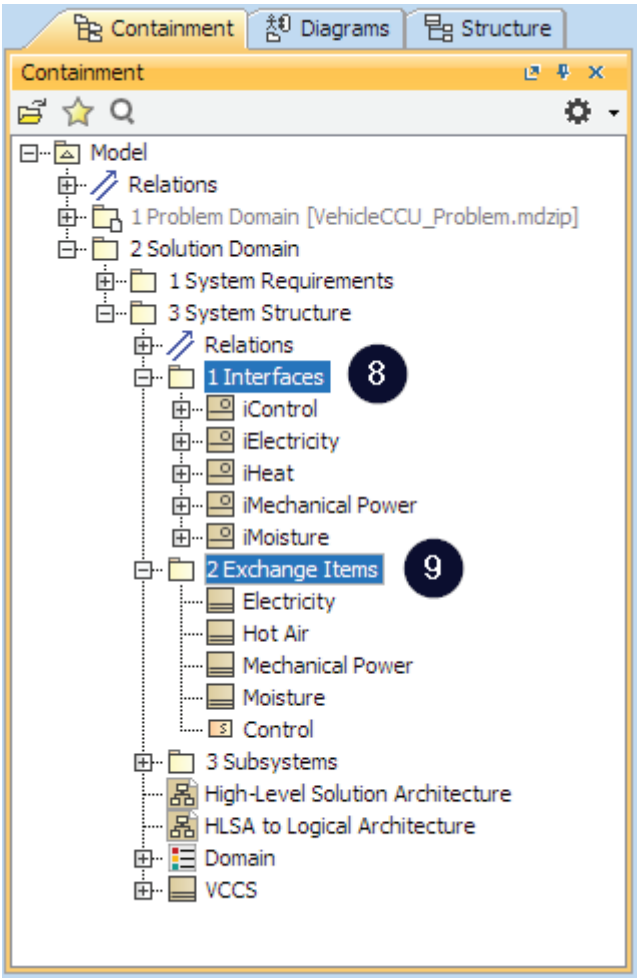


ⓘ As you can see, there are two proxy ports typed by the *iHeat* interface block. The difference between them is that the one on the left border of the *Cooling System* block is for getting the hot air to the inside of the Cooling System, and the one on the right border of that block is for dismissing the hot air to the outside of the Cooling System and even the vehicle. The latter is conjugated. To make the proxy port conjugated, right-click it and select **Is Conjugated**.



8. In the Model Browser, under the *3 Subsystems* package, select all the interface blocks and drag them to the *1 Interfaces* package you created in step 1 of the S3 initial phase tutorial. The interface blocks are moved to the *1 Interfaces* package (see the figure in step 9).

9. In the Model Browser, under the *1 Interfaces* package, select all the signals that type the flow properties of the interface blocks and drag them to the *2 Exchange Items* package you created in step 1 of the S3 initial phase tutorial. The signals are moved to the *2 Exchange Items* package.



ⓘ Note that all exchange items, except the Control signal, have been refactored to blocks. The modeling tool creates signals by default, if you type the flow properties in the above described way, though the nature of some exchange items implies to define them as blocks. This can be easily fixed by refactoring signals to blocks. Just select all the signals you want to refactor, right-click the selection, and choose **Refactor** > **Convert To** > **Block**.

In the real-world case, the systems engineer must specify interfaces of other subsystems as well. He/she might also create an ibd to specify not only interfaces, but also how these subsystems interoperate with each other.

SS3. Subsystem Structure

What is it?

Once the systems engineer identifies subsystems of the system under design (in the *HLSA* model) and allocates them to separate engineering teams, these teams can start working on the solution architecture of each subsystem. Building the solution architecture of a subsystem can start by defining the internal structure of the subsystem and specifying how its parts interoperate together, and interact with the outside of the subsystem.

To provide the solution architecture of the subsystem, the responsible engineering team needs to analyze the system requirements for that particular subsystem and the problem domain model. If there is a need, the parts of the subsystem structure can be decomposed as well. This would be the concern of the C3 cell.

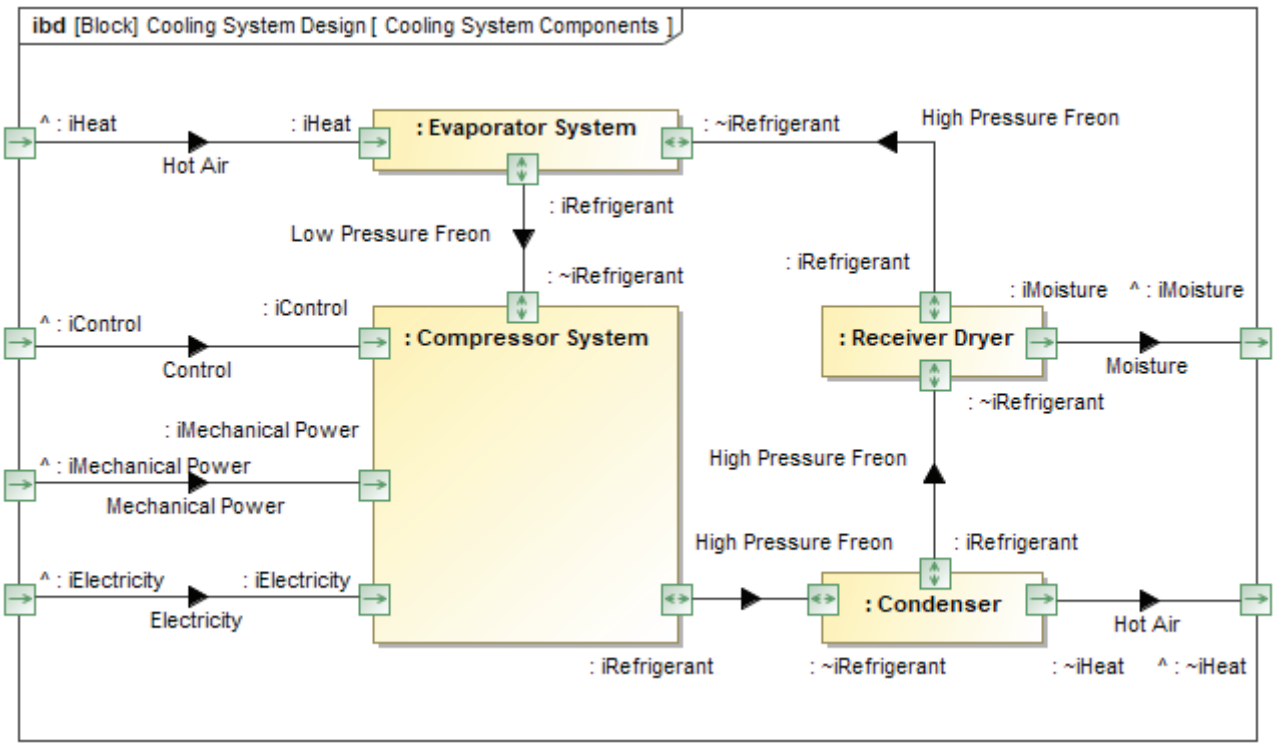
Note that the following material focuses on building the structural model of the Cooling System, only one of the subsystems of the Vehicle Climate Control System. The material also reveals how to integrate the solution into the overall solution of the system under design (see S3.final), although you should imagine that while you build the solution architecture for the Cooling System, other engineers / engineering teams work in parallel to build the structural models of the Heating System, Control System, Sensors System, and UI System.

Who is responsible?

In a typical multidisciplinary systems engineering team, the structure of the subsystem under design is built by the members of the *Architecture Team*. While the systems engineer (also known as the systems architect), the head of this team, has "a big picture view" of the system under design, other members of the *Architecture Team* (alone or in groups, also known as engineering groups) take responsibility for building the structures of the different subsystems under design.

How to model?

In the modeling tool, the parts of the subsystem can be defined by using the infrastructure of the SysML block definition diagram. Subsystems and components of the subsystem under design can be captured as blocks, and interfaces as interface blocks. For specifying how the parts of the subsystem under design interoperate together and with the outside of the subsystem, as well as what material they exchange, the infrastructure of the SysML internal block diagram can be used. The internal structure and interactions between different parts of the Cooling System in the form of the SysML internal block diagram is displayed in the following figure.



What's next?

- Once you're done with the structure of the subsystem under design, you can start thinking of and modeling the behavior of that subsystem. Therefore, you can move to the S2/SS2 cell.

Tutorial

Step 1. Creating and organizing a model for SS3

In this step, we create a separate model for the Cooling System, one of the subsystems of the VCCS defined in the *HLSA* model. In the real-world case, this model should be given to the engineering team appointed to build a solution architecture of the Cooling System (while the solution architectures of the other subsystems of the VCCS should be developed in parallel by other engineering teams).

The solution domain model of the VCCS should be used in the solution domain model of the Cooling System. This is necessary, because the imaginary engineering group must know the system requirements of the VCCS and the interfaces of the Cooling System defined in the *HLSA* model (see step 5 of the S3 initial phase tutorial) in order to follow them.

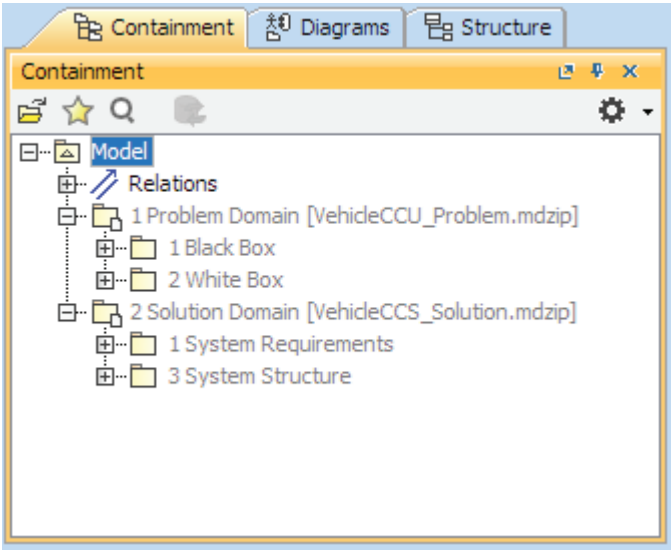
To create and organize the model for SS3

1. Start sharing the solution domain model.

   ⓘ For more information on this topic and the topics mentioned in steps 2 and 3, refer to the latest documentation of your modeling tool.

2. Create a new model (file). It can be named *Cooling System_Solution.mdzip*. This is the solution domain model of the Cooling System.

3. Use the solution domain model of the whole system under design as *read-only* in the solution domain model of the Cooling System. The problem domain model is automatically used as well.



   ⓘ Names in grey indicate that elements cannot be modified. This is because the problem domain model is used in the solution domain model as *read-only*.

4. Right-click the *Model* package (this is the default name of the root package) and select **Create Element**.

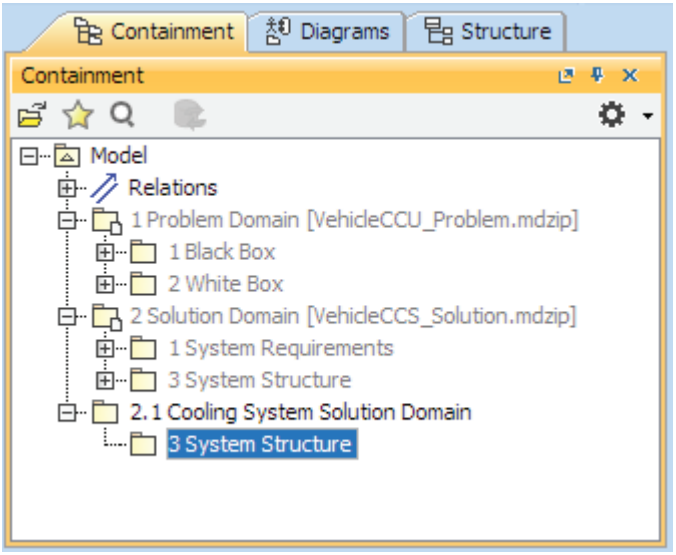
5. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.

6. Type *2.1 Cooling System Solution Domain* to specify the name of the new package and press Enter (see the figure in step 9).

   ⓘ Packages of other subsystems under design, like the Heating System and Sensors System to name a few, should include 2.2, 2.3, and so on in their names.

7. Right-click the *2.1 Cooling System Solution Domain* package and select **Create Element**.

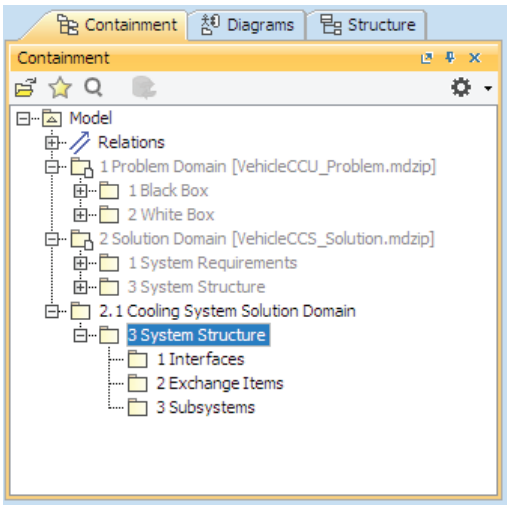
8. Repeat step 5.
9. Type *3 System Structure* to specify the name of the new package and press Enter.



For the purpose of keeping the inner structure of the *3 System Structure* package well-organized, you need to create a few more packages. These are:

- *1 Interfaces*
- *2 Exchange Items*
- *3 Subsystems*

When you create them on your own (by following the previous procedure), the Model Browser of your solution domain model should look the same as displayed in the following figure.



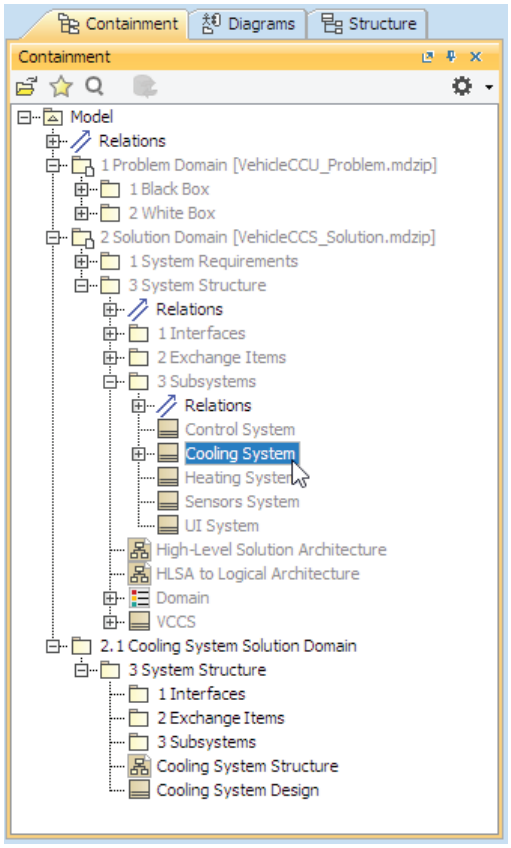
Step 2. Getting ready for modeling the structure of the Cooling System

Once the model for capturing the solution architecture of the Cooling System is created, there is one more thing to do before giving the model to the appointed engineering team. You, as the systems engineer, must establish the inheritance between two concepts: the *Cooling System* block in the *HLSA* model (as the super-type) and the one representing the same subsystem in the solution domain model of that subsystem (as the sub-type). The inheritance enables the systems engineering team to learn all the interfaces of the subsystem under design determined by the systems engineer in the *HLSA* model.

The inheritance is captured in the model as the generalization relationship between the abovementioned blocks. It can be simply created by utilizing the infrastructure of a bdd. The bdd can be created under the *3 System Structure* package within the solution domain model of the Cooling System (created in step 1 of the SS3 initial phase tutorial).

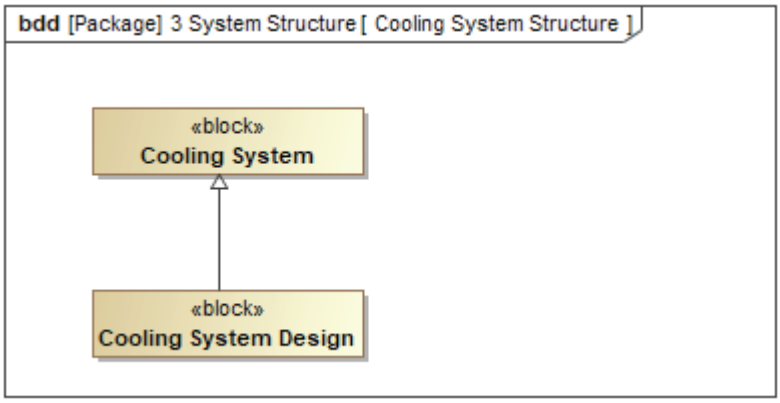
To get ready for modeling the structure of the Cooling System

1. Create the bdd:
   a. Right-click the *3 System Structure* package and select **Create Diagram**.
   b. In the search box, type bdd, the acronym for SysML block definition diagram, and then press Enter. The diagram is created.
   c. Type *Cooling System Structure* to specify the name of the new diagram and press Enter again.
2. Create the block of the subsystem under design:
   a. Click the **Block** button on the diagram palette and then click on the diagram pane.
   b. Type *Cooling System Design* to name the block and press Enter.
3. Display the *Cooling System* block from the *HLSA* model on the diagram:
   a. Select the *Cooling System* block in the Model Browser.



   b. Drag the selection to the diagram pane. The shape of the *Cooling System* block is created on the diagram.
4. Draw a generalization between the *Cooling System* block (super-type) and the *Cooling System Design* block (sub-type):
   a. Click the **Generalization** button on the diagram palette.
   b. Select the shape of the *Cooling System Design* block.

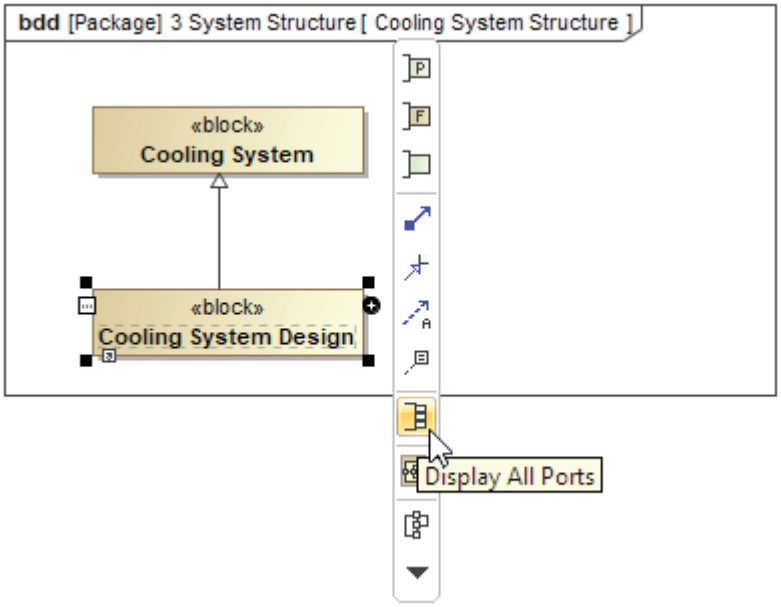
c. Select the shape of the *Cooling System* block.



5. Display the inherited interfaces of the *Cooling System Design* block on its shape:

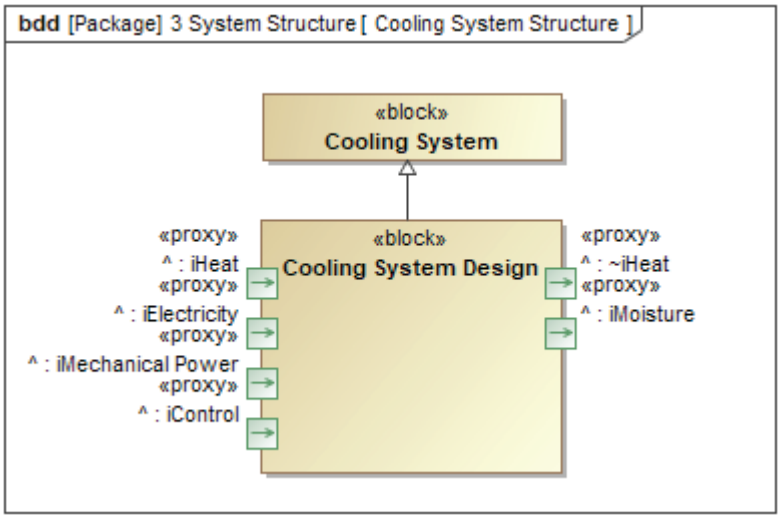
   a. Select the shape of the *Cooling System Design* block on the diagram pane.

   b. Click the Display All Ports button on its smart manipulator toolbar.



The inherited interfaces are displayed. The caret symbol ("**^**") indicates them as inherited.

> ⓘ As you can see, port names are hidden on the diagram. As these names convey no important information, hiding them is a good practice to enhance the diagram's readability. To hide the name of the port, right-click its shape on the diagram pane and unselect the **Show Name** check box. If you want to set this option for all ports



Step 3. Capturing components of the Cooling System

Let's say the engineering team came to the decision that the Cooling System should be composed of:

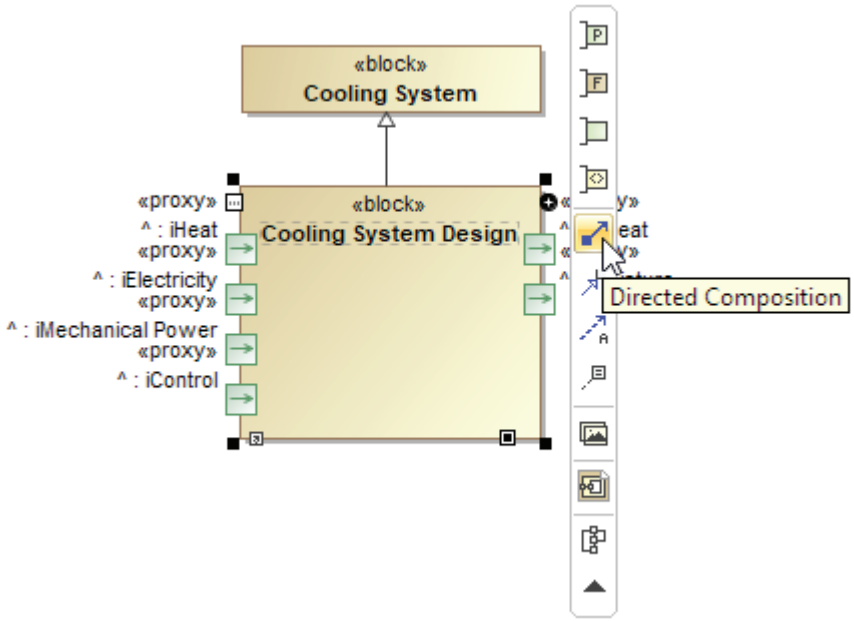
- Evaporator System
- Compressor System
- Condenser
- Receiver Dryer

Note that some components of the Cooling System have the keyword *System* in their names. This means they have their own components that can be specified in the C3 cell. All the items can be captured using either bdd or ibd, though we use the bdd *Cooling System Structure* created in step 2 of the SS3 tutorial. The ibd is useful, when you need to represent interactions between the components.

To capture the components of the Cooling System

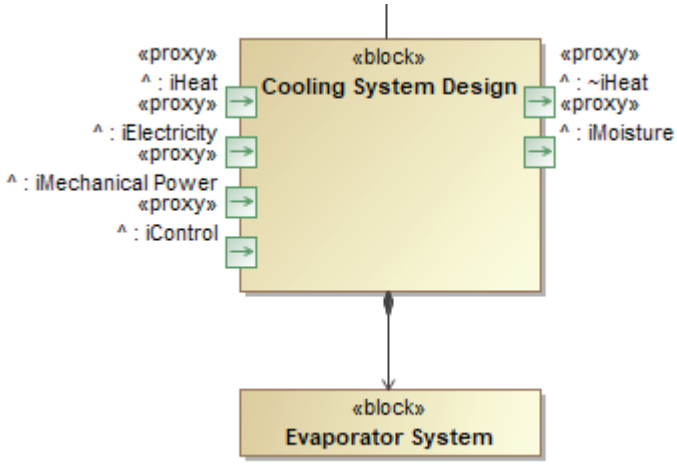
1. Open the bdd Cooling System Structure, created in step 2 of the SS3 tutorial, if not opened yet.

2. Select the *Cooling System Design* block, click the Directed Composition button on its smart manipulator toolbar (see the following figure), and then click a free space on the diagram pane.

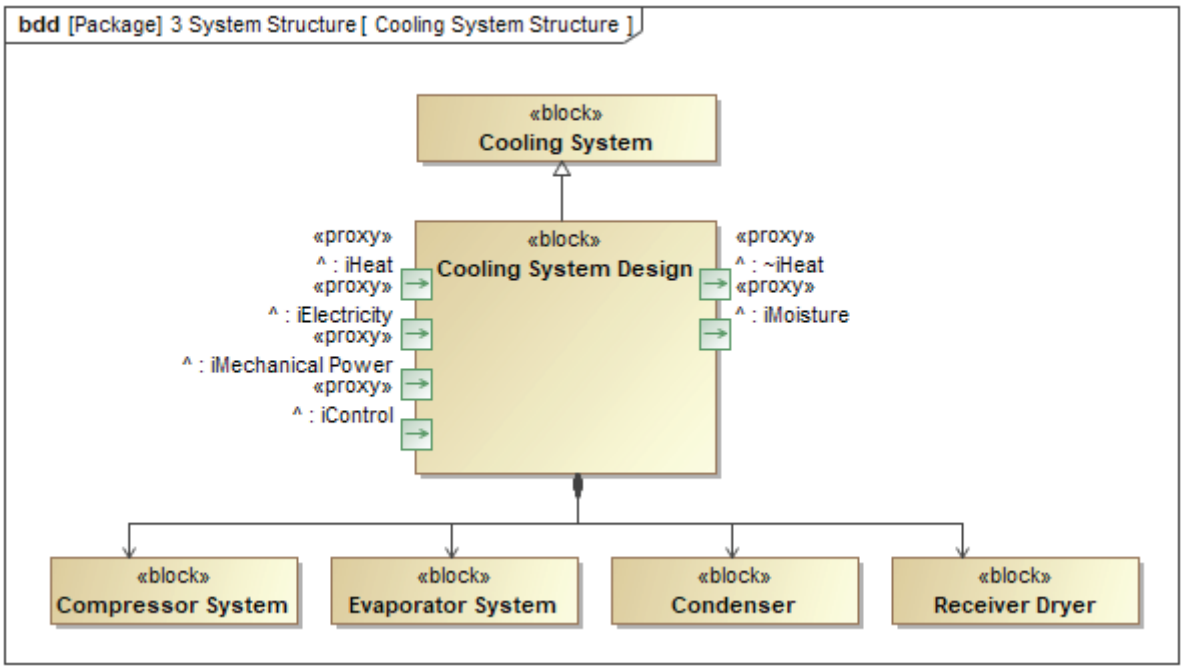


Another block is created in the Model Browser and its shape is selected on the diagram pane.

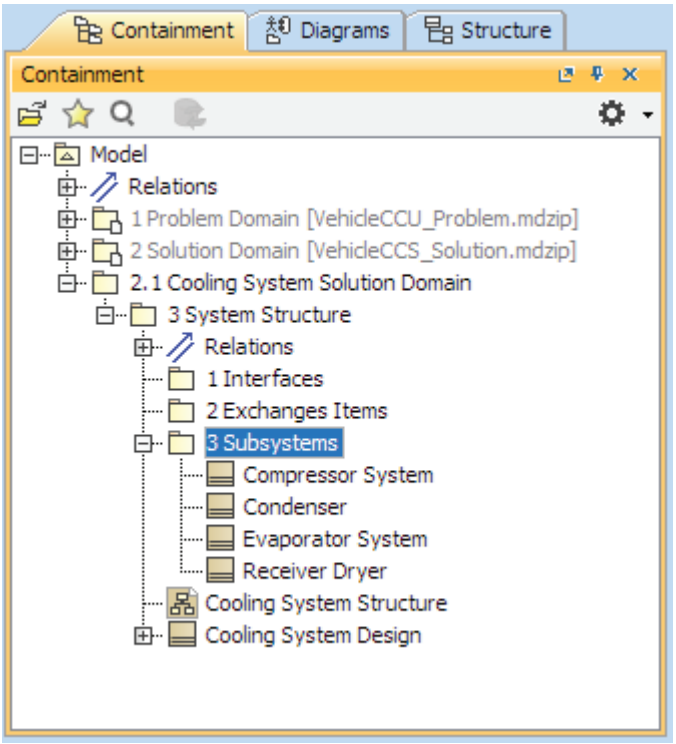
3. Type *Evaporator System* directly on the selected shape to specify the name of the appropriate component of the Cooling System, and press Enter.



4. Repeat steps 2 and 3 as many times as you need to create the *Compressor System, Condenser*, and *Receiver Dryer* blocks.



5. In the Model Browser, select all the blocks o f subsystems/components and drag them to the *3 Subsystems* package you created in step 1 of the S33 tutorial. The blocks are moved to the *3 Subsystems* package.



Note that a directed composition represents the usage of one block in another. The usage is captured in the model as a part property of the enclosing block. For example, the *Cooling System Design* block is an enclosing block which currently has four part properties, each representing a usage of a single block, such as *Condenser* or *Evaporator System*. Just like in the logical architecture model (see step 6 of the W3 tutorial), these part properties don't need names. Part properties can have names, though, when there is a need to convey how the component is used in the subsystem context. For example,

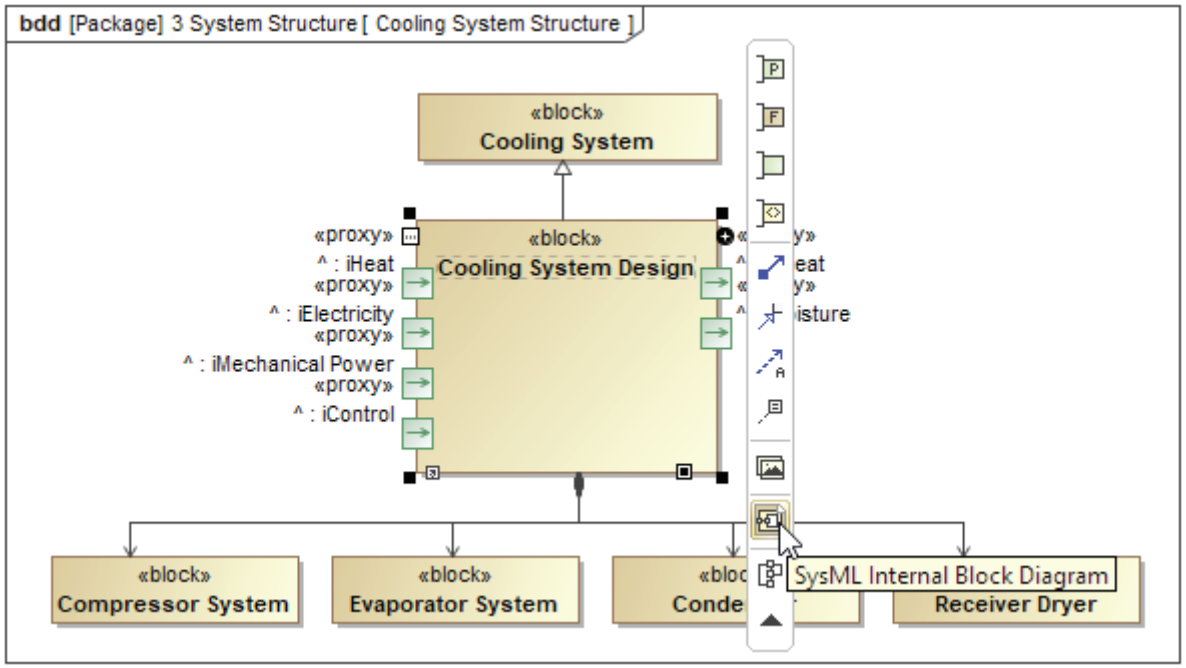
when you want to specify that the vehicle has one front-left wheel and one front-right wheel, you need to define the *Front Wheel* block in your model and then create two part properties typed by that block in the vehicle context. One part property can be named *left* and the other *right*.

Step 4. Creating an ibd for specifying interactions

While a bdd works best to capture the components, an ibd is helpful when you need to specify interactions between these components. That's why we need to create an ibd for the *Cooling System Design* block. To get started specifying the interactions, you need to represent all proxy ports and directly used components of the Cooling System in that diagram. They all have been captured using the bdd *Cooling System Structure*.

To create an ibd for specifying interactions of the Cooling System

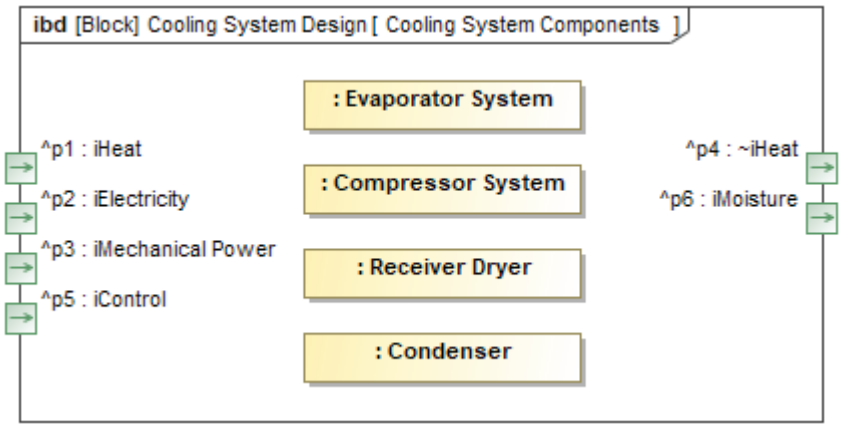
1. In the bdd *Cooling System Structure*, select the *Cooling System Design* block and click the SysML Internal Block Diagram button on its smart manipulator toolbar.



The ibd is created and the **Display Parts/Ports** dialog opens on the top of it.

2. Make sure all the items (both proxy ports and part properties!) are selected and click **OK**. The dialog is closed, and you can see:
   - The ibd with its frame representing the boundaries of the Cooling System.
   - Proxy ports representing the interfaces of the Cooling System.
   - Part properties representing all the direct components of the Cooling System.

3. In the Model Browser, where the newly created diagram is automatically selected in the edit mode, type *Cooling System Components* to specify its name and press Enter again.



Step 5. Specifying interactions between the Cooling System and the outside

In this step, we focus on specifying interactions between subsystems/components of the Cooling System and the outside of it; that is, other subsystems of the VCCS or even other systems of the vehicle. Interactions can be specified as connectors established via compatible proxy ports. Information, resources, messages, and so forth exchanged via the connectors can be specified as items flowing over these connectors.

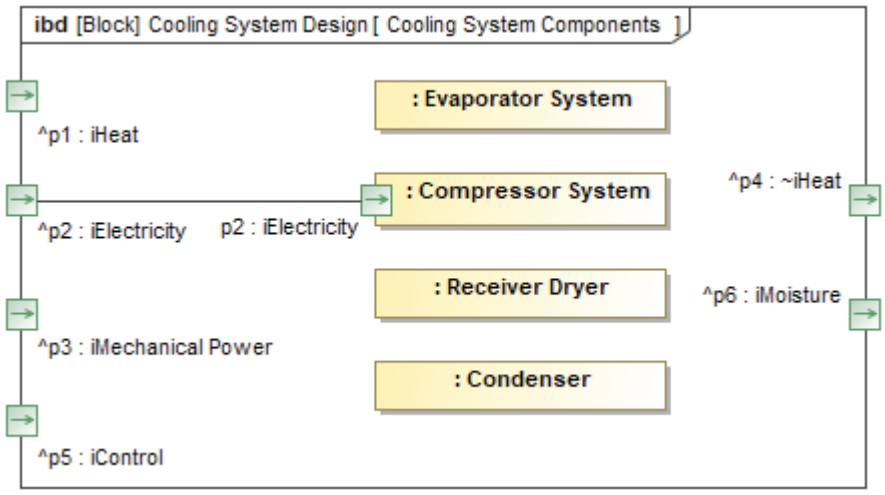
Let's say the engineering team identified the following interactions:

- Compressor System consumes electricity and mechanical power (from other systems of the entire vehicle, like generator and engine) and accepts control signals (from the Control System of the Vehicle Climate Control System)
- Evaporator System takes the heat from the vehicle cabin (actually, by cooling down the air)
- Condenser System provides the heat to the outside of the vehicle (actually, by heating up the air)
- Receiver Dryer removes from the Cooling System the moisture separated from the Freon

To specify that the Compressor System consumes electricity from outside the Cooling System

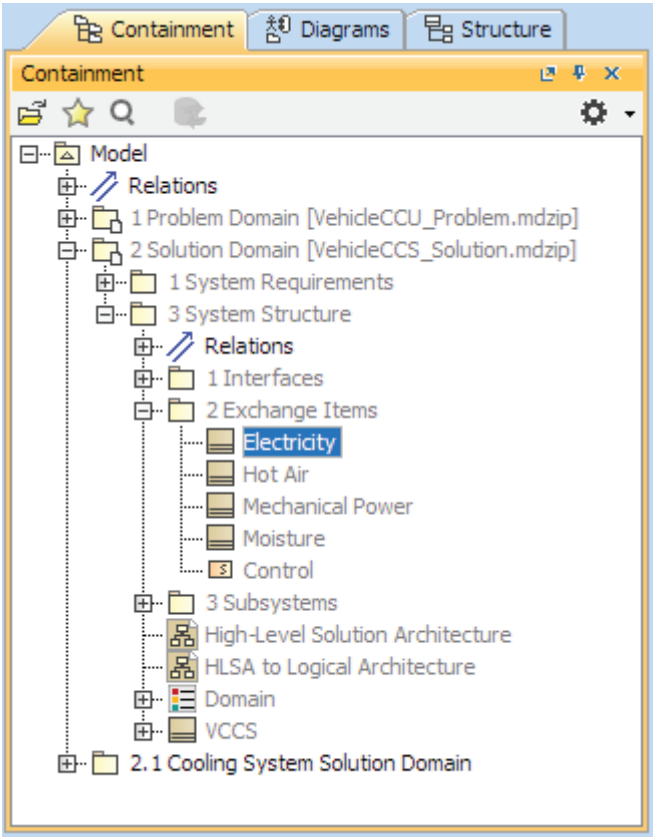
1. Select the *p2* proxy port (typed by the *iElectricity* interface block) on the diagram frame and click the Connector button on its smart manipulator toolbar.
2. Click the shape of the part property typed by the *Compressor System* block.

3. Select **New Proxy Port**. A compatible proxy port is created for the part property, and the connector is established between the diagram frame and the selected part property.



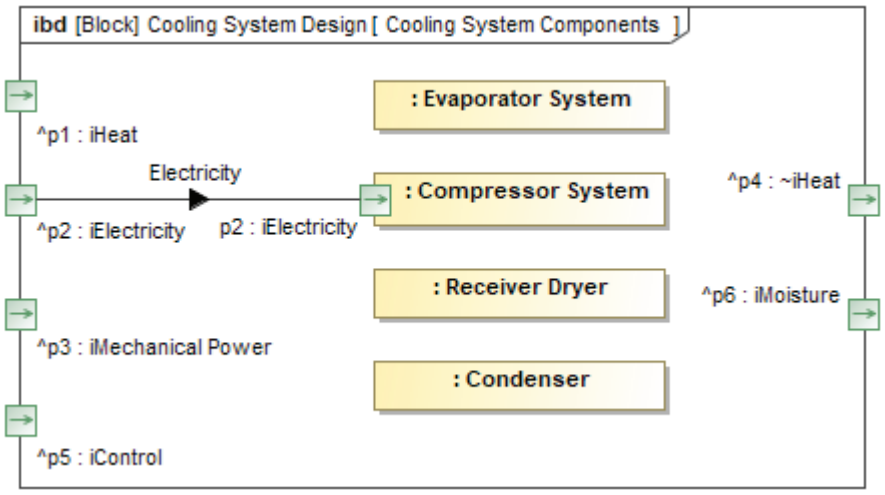
4. In the Model Browser, expand the contents of the following read-only packages: *2 Solution Domain* > *3 System Structure* > *2 Exchange Items*.

5. Select the *Electricity* block (see the following figure) and drag it to the newly created connector on the open diagram pane.

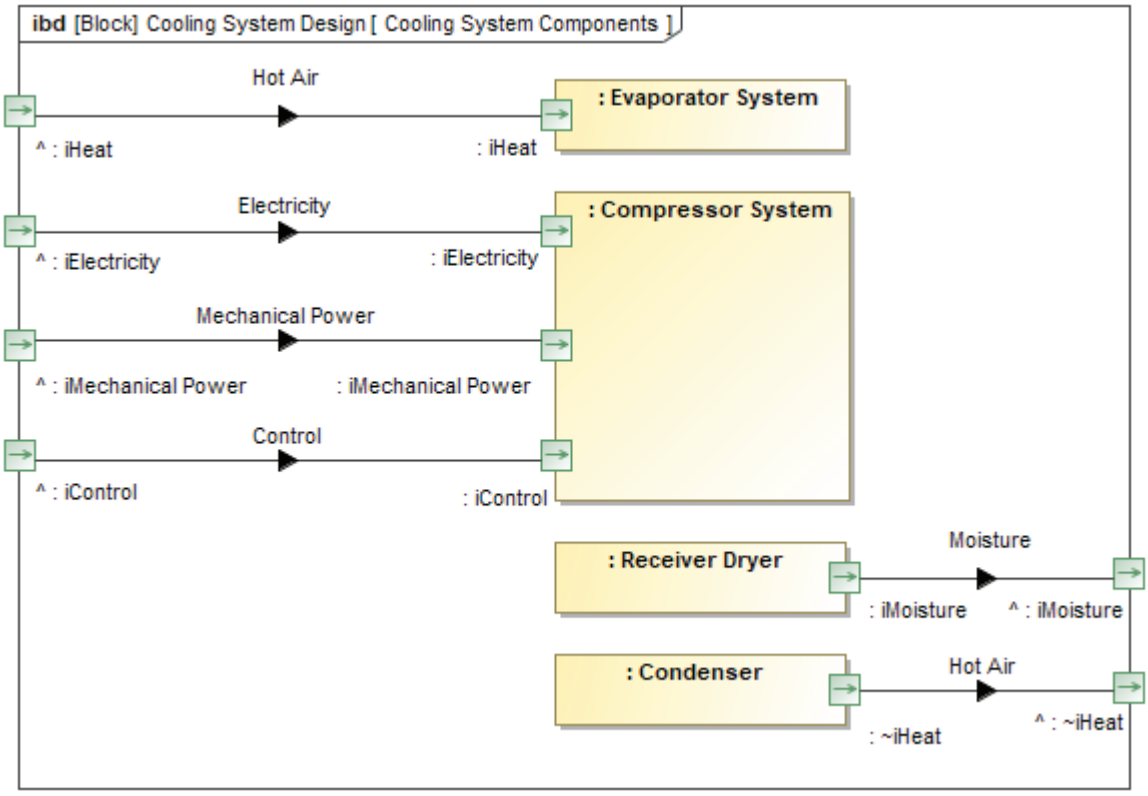


ⓘ According to SysML, the element you want to specify as the item flowing via the connector must be the type of at least one of the flow properties owned by the interface block that types the connected proxy ports.

6. Don’t make any changes in the open dialog; click **OK**. The Electricity block is specified as the item flowing over the newly created connector.



In the same way, specify the rest of the previously described interactions. Once you’re done, your ibd should be similar to the one in the following figure.



ⓘ As you can see, port names are hidden on the diagram. As these names convey no important information, hiding them is a good practice to enhance the diagram’s readability. To hide the name of the port, right-click its shape on the diagram pane and unselect the **Show Name** check box. If you want to set this option for all ports at once, select the shape of any port while holding down the Alt key and proceed with the same actions as for a single port.

Step 6. Specifying interactions within the Cooling System

In this step, we proceed with specifying interactions – this time, between different components of the Cooling System. As in the previous step, interactions can be specified as connectors established via compatible proxy ports. Information, resources, and so forth exchanged via the connectors can be specified as item flows.

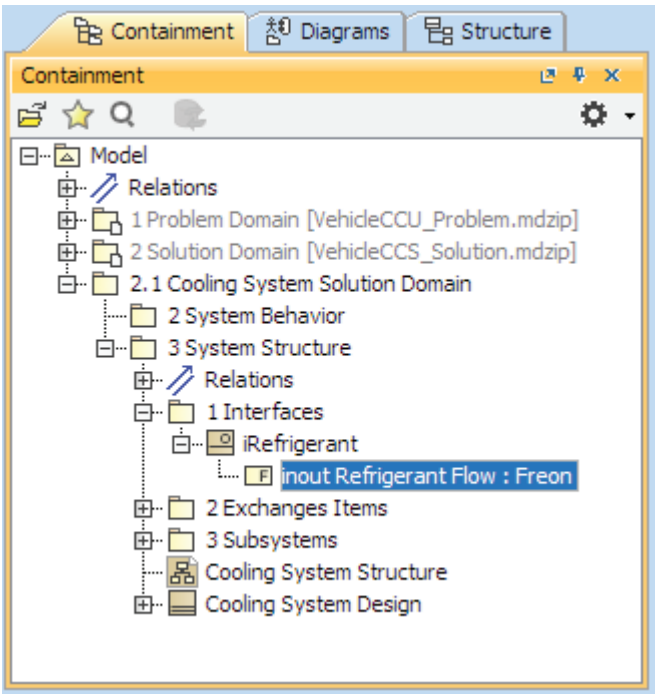
Let's say the engineering team identified the following interactions:

- Compressor System takes low pressure Freon from the Evaporator System
- Compressor System transforms Freon from low pressure to high pressure gas (almost liquid)
- Condenser takes the high pressure Freon from the Compressor System and provides it to the Evaporator System, through the Receiver Dryer
- Evaporator System transforms the Freon from high pressure to low pressure gas and provides it to the Compressor System

The list above determines one more interface of the Cooling System. It is for exchanging Freon or, generally speaking, Refrigerant. It can be captured in the model as a new interface block.

To capture the interface block for exchanging Refrigerant

1. Right-click the *1 Interfaces* package that you created in step 1 of the SS3 tutorial and select **Create Element**.
2. In the search box, type *ib*, where i stands for *interface* and *b* for *block*, and press Enter.
3. Type iRefrigerant to specify the name of the new interface block and press Enter.
4. Right-click the interface block and select **Create Element**.
5. In the search box, type *fp*, where *f* stands for *flow* and *p* for *property*, and press Enter.
6. Leave *inout*, the default direction of the flow property, as it is; type *Refrigerant Flow:Freon to* specify its name and type, and then press Enter. The flow property is created together with the *Freon* signal as its type.



As you can see, Freon was automatically captured in the model as a signal, which is not correct, as the nature of Freon implies that it should be defined as a block. This can be easily fixed by refactoring the *Freon* signal to a block.

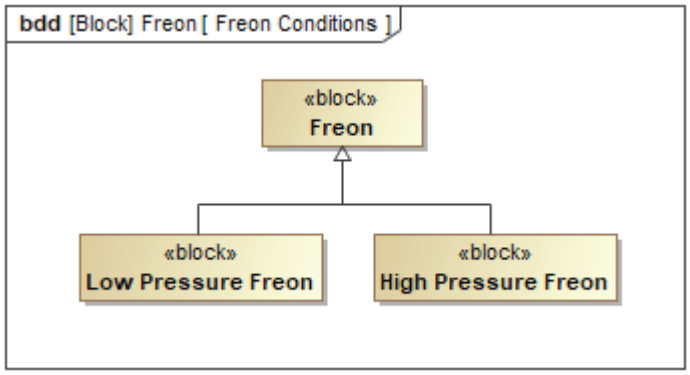
To refactor the *Freon* signal to a block

1. In the Model Browser, right-click the Freon signal and select **Refactor** > **Convert To** > **Block**.
2. Click **OK** to remove all the properties that are incompatible between these two types.

The next thing you need for specifying the above described interactions is the relevant exchange items. Since Freon can be either low pressure gas or high pressure gas, you need to define both of these conditions in your model. Each condition can be captured as a sub-type of the *Freon* block. For creating sub-types, the infrastructure of the bdd can be used. Also note that the *Freon* block and both of its subtypes should be stored in the *2 Exchange Items* package.

To capture the conditions of Freon

1. In the Model Browser, select the *Freon* block and drag it to the *2 Exchange Items* package that you created in step 1 of the SS3 tutorial. The *Freon* block is moved to that package.
2. Right-click the *2 Exchange Items* package and select **Create Diagram**.
3. In the search box, type *bdd*, the acronym for SysML block definition diagram, and then press Enter. The diagram is created.
4. Type *Freon Conditions* to specify the diagram name and press Enter again.
5. Drag the *Freon* block to the diagram pane. The shape of the *Freon* block is created on the diagram.
6. Click the Generalization button on its smart manipulator toolbar.
7. Click an empty place on the diagram pane.
8. Type *Low Pressure Freon* directly on the new shape to name the block and press Enter.
9. Select the Freon block again and follow steps 6 to 8 to create the *High Pressure Freon* block.



Now you're ready to specify the interactions between the components of the Cooling System (identified at the beginning of this step).

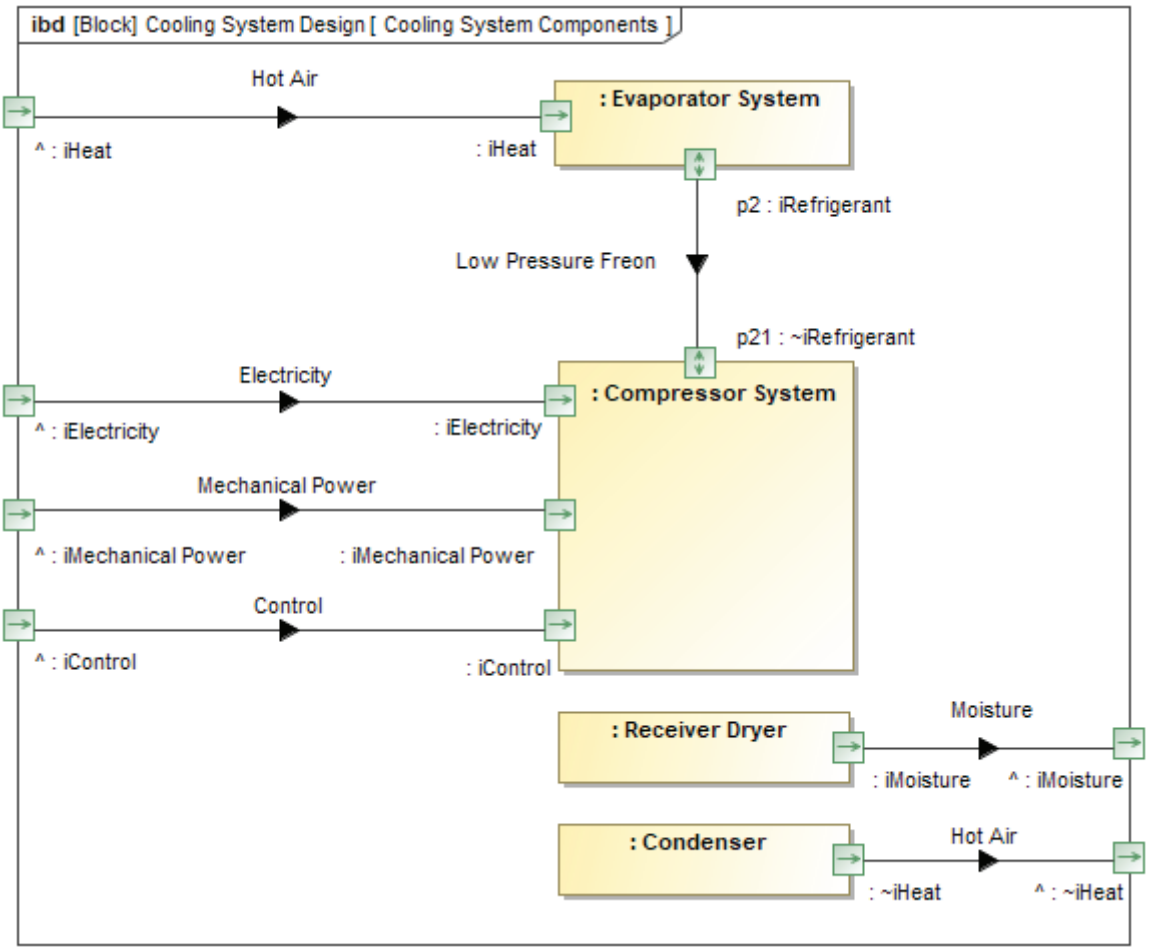
To specify that the Compressor System takes low pressure Freon from the Evaporator System

1. Make sure the Type Selection Mode is enabled in the *Cooling System Components* diagram. Otherwise, creation of a proxy port doesn't trigger selection of the interface block to type that proxy port.
2. Select the shape of the part property typed by the *Evaporator System* block and click the Proxy Port button on its smart manipulator toolbar. A new proxy port appears on the border of the part property shape.
3. Type *ir* to find the *iRefrigerant* interface block in the **Select Type** list, press the Down Arrow key to select it, and then press Enter.

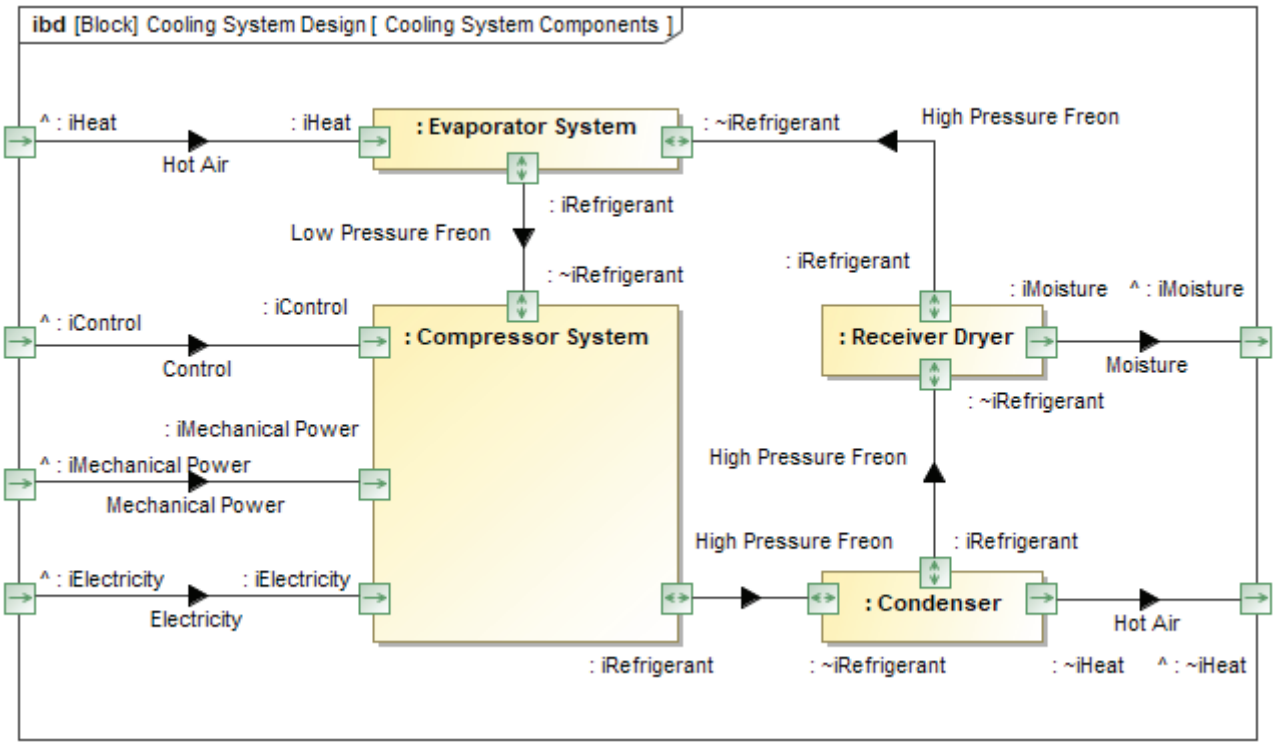
4. Click the Connector button on the smart manipulator toolbar of the newly created proxy port.
5. Click the shape of the part property typed by the *Compressor System* block.
6. Select **New Proxy Port**. A compatible proxy port is created for that part property, and the connector between the couple of part properties is established.
7. In the Model Browser, expand the contents of the *2 Exchange Items* package (if it is not expanded yet), select the *Low Pressure Freon* block, and drag it to the newly created connector.

> ⓘ According to SysML, the element you want to specify as the item flowing via the connector must be the type of at least one of the flow properties owned by the interface block that types the connected proxy ports. This also works with sub-types. That's why not only the *Freon* block, but any of its types (in this case, the *Low Pressure Freon* block), can be assigned to the connector.

8. Don't make any changes in the open dialog; click **OK**. The *Low Pressure Freon* block is specified as the item flowing via the newly created connector.



In the same way, specify the rest of the previously described interactions. Once you're done, your ibd should be similar to the one in the following figure.



S2/SS2. System/Subsystem Behavior

What is it?

The behavior of the system under design integrates the behavior of its subsystems, identified in the *HLSA* model (see S3.initial). So first of all, the behavior of the subsystems under design must be specified. Separate engineering teams, each responsible for the solution architecture of a single subsystem, work in parallel for this purpose. Behavior models can be relatively abstract and should satisfy functional system requirements (see S1.initial). Using the capabilities of the modeling tool, these models can be executed against test cases to give the verdict on whether these functional requirements are satisfied or not.

Afterwards, the behavior models of all the subsystems under design are integrated into a single one – the behavior model of the entire system. The main purpose of the behavior model integration is to check how these subsystems communicate with each other by sending signals to one another via the compatible ports defined in S3. Therefore, the main task of the systems engineer is to test whether the signal sent from one subsystem can be accepted by another. Again, he/she can utilize the simulation capability of the modeling tool for this purpose.

Note that the following material focuses on building the behavioral model of the Cooling System, only one of the subsystems of the Vehicle Climate Control System. However, you should image that while you build the behavioral models for the Cooling System, other engineers / engineering teams work in parallel to build the solution architectures of the Heating System, Control System, Sensors System, and UI System.

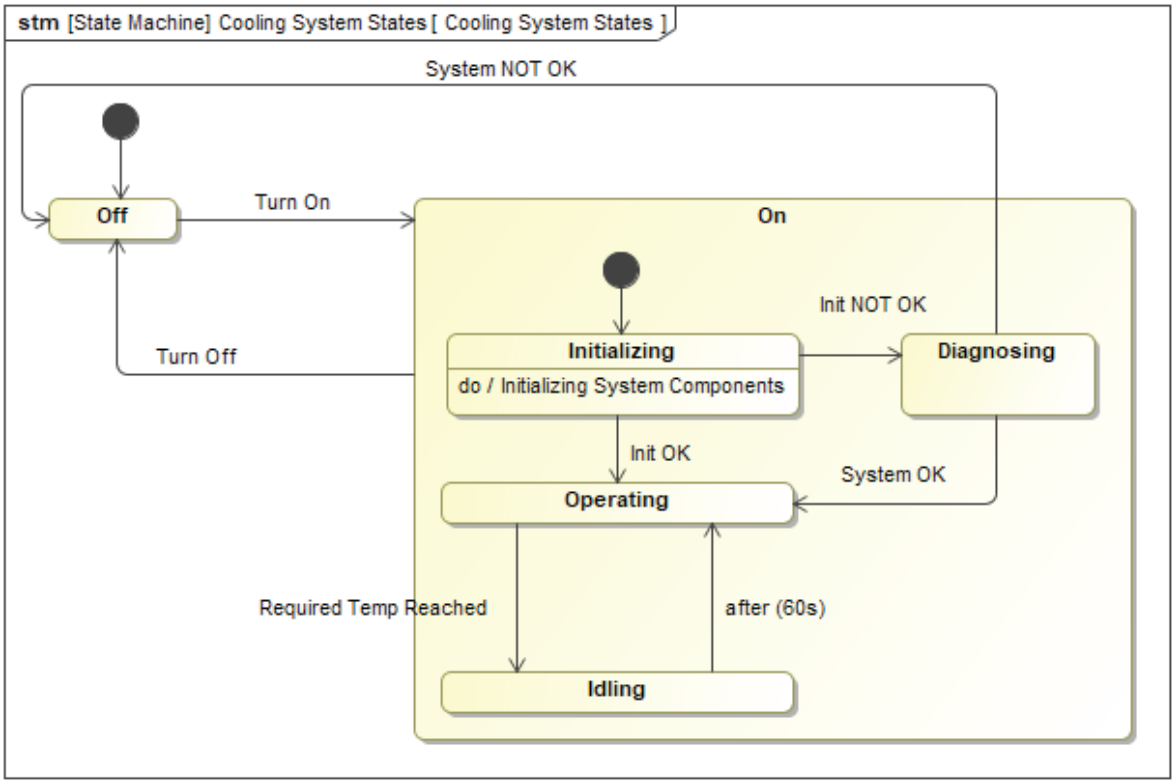
Who is responsible?

In a typical multidisciplinary systems engineering team, the behavior of the system/subsystem under design is produced by the *Architecture Team*. Engineering teams build isolated behavioral models of each subsystem under design. The systems engineer afterwards integrates them into the whole.

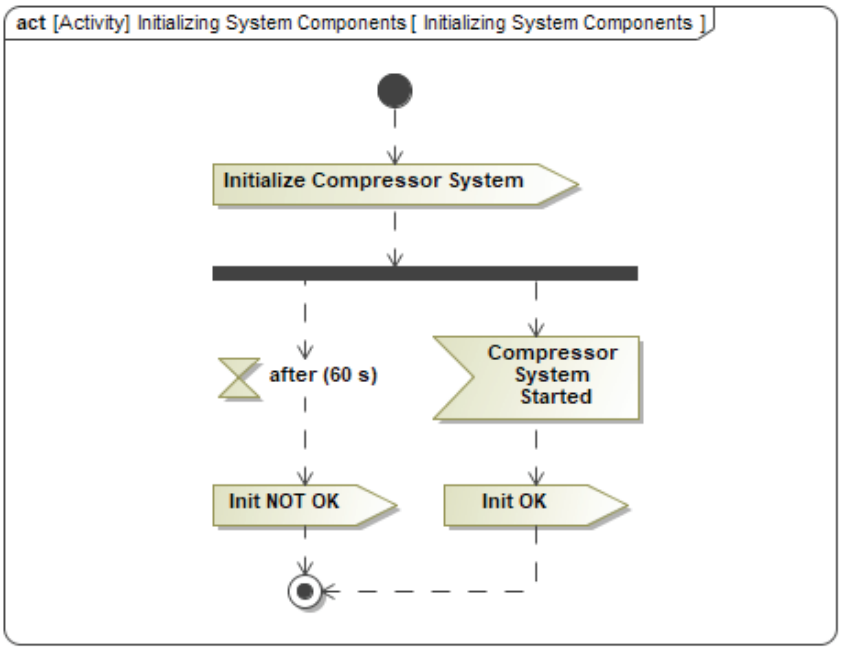
How to model?

The behavior model of the system or subsystem under design can be captured by utilizing the infrastructure of the SysML state machine and activity or sequence diagrams in combination. The former allows for capturing states of the subsystem under design and transitions between them in response to event occurrences over time. The latter should be used to specify the *entry, do*, and *exit* behaviors of these states or transition effects.

The following figure displays the SysML state machine diagram, identifying the main states of the Cooling System. Transitions between states can be triggered by various types of events. As you can see, it can be a signal event, such as *Turn On* between the *Off* and *On* states, or a time event, such as *after (60s)* between the *Idling* and *Operating* states.



States can have one or more internal behaviors that are specified in the form of the SysML activity diagram created somewhere in the model. As you can see, the *Initializing* state has the *Initializing System Components* activity specified as the do behavior. The *Initializing System Components* activity diagram is displayed as follows.



Having the behavior models of each subsystem under design, you, as the systems engineer, can create a SysML activity diagram to specify the signal sent from the Cooling System to, for example, *Control System* via compatible ports. The send signal action should specify the port that sends the signal, not the one that accepts it.

What's next

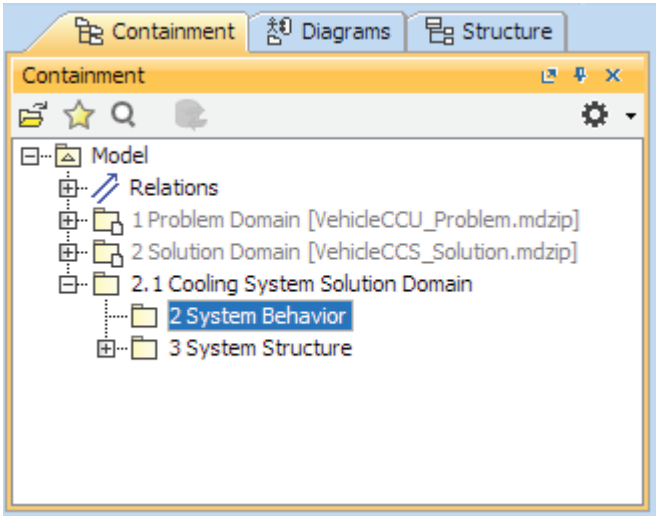
- Once you have subsystems behavior, you can start integrating the solution architectures of all subsystems into the whole; that is, move to S3.final.

Tutorial

Step 1. Organizing the model for SS2

The states of the Cooling System can be captured in the solution domain model of that subsystem, the one you created in step 1 of the SS3 tutorial. According to SysML, states can only be stored under the block whose behavior they represent. In this case, it is the *Cooling System Design* block. Therefore, you don't need to create any additional packages for storing states in the model. They should appear directly under the *Cooling System Design* block, which is stored within the *3 System Structure* package.

However, if you need to specify some behavior that the Cooling System performs while being in one or another state or in transition from one state to another, you should specify that behavior in a separate package. Following the design of the MagicGrid framework, this package should be stored under the structure of packages displayed in the following figure.



To organize the model for SS2

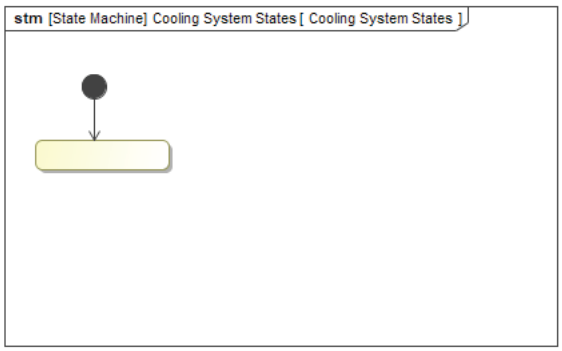
1. Open the solution domain model of the Cooling System (the *Cooling System_Solution.mdzip* file) you created in step 1 of the SS3 tutorial, if not opened yet.
2. Right-click the *2.1 Cooling System Solution Domain* package and select **Create Element**.
3. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.
4. Type *2 System Behavior* to specify the name of the new package and press Enter.

Step 2. Creating a diagram for capturing Cooling System states

The SysML state machine diagram that describes the states of the Cooling System should be created directly under the block which captures that subsystem under design.

To create a SysML state machine diagram for capturing the Cooling System states

1. Open the solution domain model of the Cooling System you created in step 1 of the SS3 tutorial, if not opened yet.
2. In the Model Browser, select the *Cooling System Design* block. For this, use the quick find capability:
   a. Press Ctrl + Alt + F. The **Quick Find** dialog opens.
   b. Type *cooling s*.
   c. When you see the *Cooling System Design* block selected in the search results list below, press Enter. The *Cooling System Design* block is selected in the Model Browser.
3. Create the SysML state machine for the *Cooling System Design* block:
   a. Right-click the *Cooling System Design* block and select **Create Diagram**.
   b. In the search box, type *smd*, the acronym for the SysML state machine diagram, and then press Enter. The diagram is created with the initial state and an unnamed one to jump-start the states definition.



   c. Type *Cooling System States* to specify the name of the new diagram and press Enter again.

Step 3. Capturing states of the Cooling System

Let's say the engineering team working on the solution architecture of the Cooling System has identified the following set of states in which the Cooling System can exist during the VCCS operation:

- Off
- On:
  - Initializing
  - Diagnosing
  - Operating
  - Idling

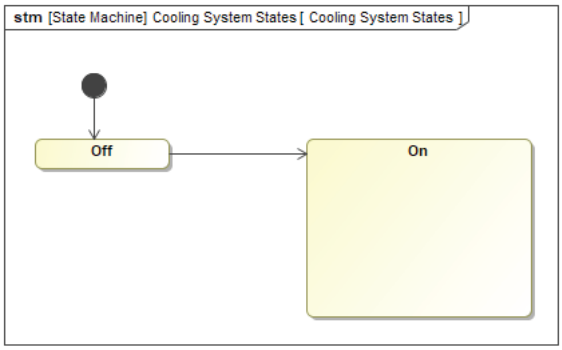
The following rules describe transitions between those states:

- The first state of the subsystem is *Off*.
- The subsystem can move from the *Off* state to the *On* state and back.
- Being in the *On* state, the subsystem goes to the *Initializing* state first.
- From the *Initializing* state, the subsystem can move either to the *Operating* state or to the *Diagnosing* state.
- From the *Operating* state, the subsystem can move to the *Idling* state and back.
- From the *Diagnosing* state, the subsystem can switch either to the *Operating* state or to the *Off* state.

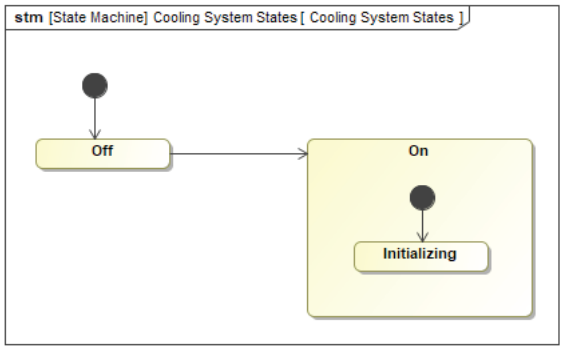
Event occurrences that trigger transitions between states are discussed later in this tutorial. This step mainly focuses on creating states and transitions between them.

To specify states of the Cooling System

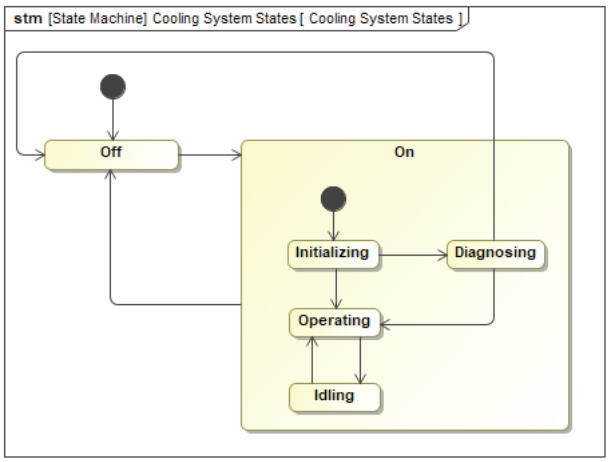
1. Open the *Cooling System States* diagram, if not opened yet. The diagram already contains the initial state and an unnamed state.
2. Select the shape of the unnamed state and then click somewhere in the middle of that shape. The shape switches to the name edit mode.
3. Type *Off* to specify the name and press Enter.
4. Select the shape of the *Off* state and click the Transition button on its smart manipulator toolbar.
5. Click an empty place on the diagram pane. Another state is created.
6. Type *On* directly on the shape of that state and press Enter.
7. Drag the corner of the newly created shape to enlarge it. This is necessary for creating internal states within it.



8. Click the **Initial** button on the diagram palette and move the mouse over the shape of the *On* state.
9. When you see the blue border around the shape of the *On* state, click it. The initial state is created within the shape of the *On* state, and the *On* state automatically becomes composite.
10. Select the shape of the initial state and click the Transition button on its smart manipulator toolbar.
11. Right-click an empty place on the shape of the composite state and select **State**. Another internal state of that composite state is created.
12. Type *Initializing* directly on the shape of that state and press Enter.



13. Create the rest of the states from the list above following the given rules of transition. When you're done, your state machine diagram should be very similar to the one in the figure below.

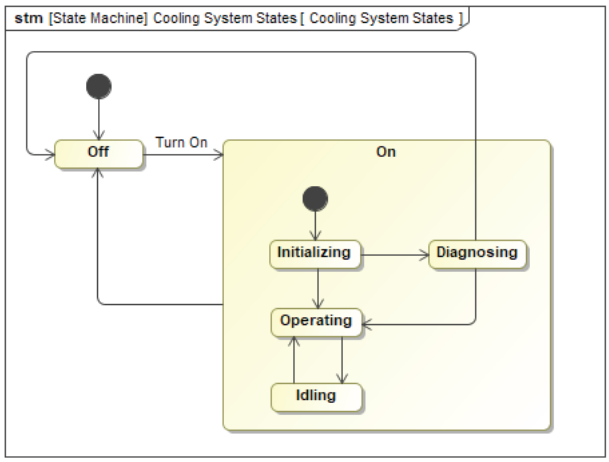


Step 4. Specifying event occurrences on transitions

Transitions between states can be triggered by various types of events, such as time, change, and signal events, to name a few. The majority of the transitions between states of the Cooling System are triggered by signal events. For example, the Cooling System moves from the *Off* to the *On* state after it receives the *Turn On* signal; it moves back to the *Off* state after the *Turn Off* signal is received.

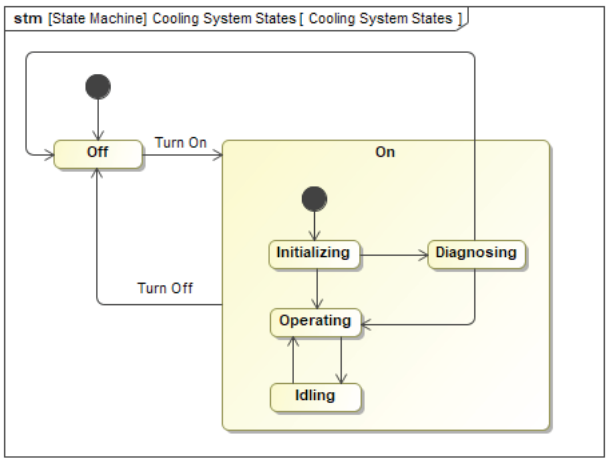
To specify signal events on transitions between the *On* and *Off* states of the Cooling System

1. Select the transition from the *On* to *Off* state.
2. Type *Turn On* directly on the path and press Enter. The *Turn On* signal is created in the Model Browser and automatically assigned as the signal event to the transition from the *Off* state to the *On* state.

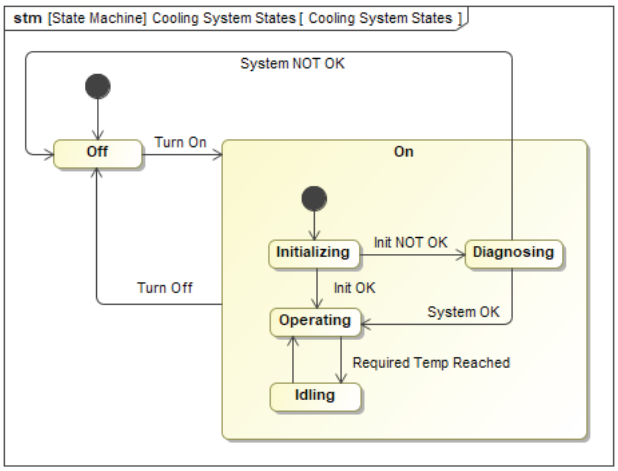


3. Select the transition from the *Off* to *On* state.

4. Type *Turn Off* directly on the path and press Enter. The *Turn Off* signal is created in the Model Browser and automatically assigned as the signal event to the transition from the *On* state to the *Off* state.



The *Turn On* and *Turn Off* signals, as well as subsequently created signals (see the figure below), should be stored in a separate package. This is discussed later in this tutorial.

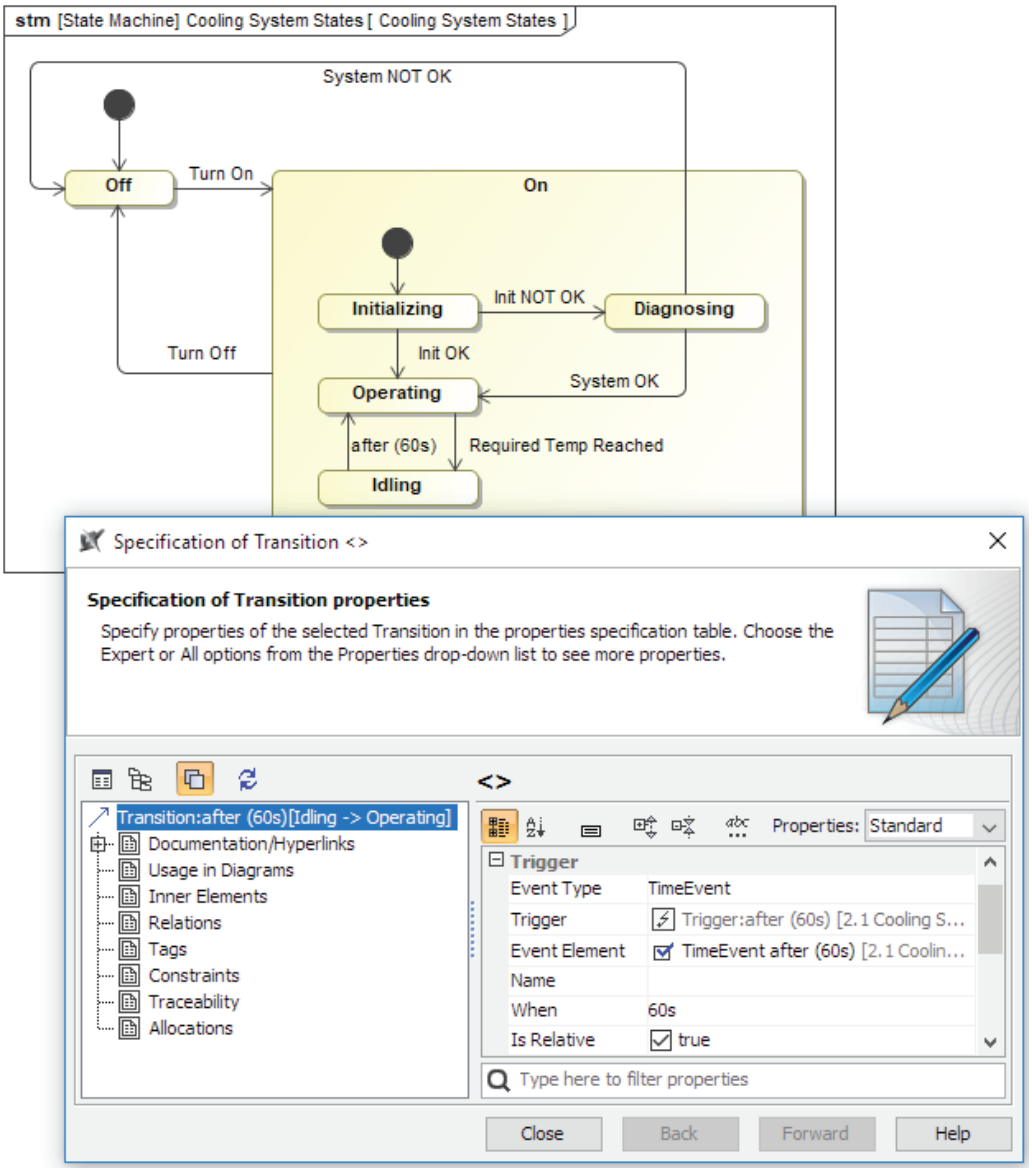


Transition from the *Idling* state to the *Operating* state is triggered by another type of event – a relative time event. It allows the modeler to specify that after 60 seconds of being in the *Idling* state, the Cooling System moves back to the *Operating* state.

To specify the time event in transition between the *Idling* and *Operating* state

1. Select the transition from the *Idling* to *Operating* state.

2. Type *after (60s)* directly on the path and press Enter. The relative time event is created on the transition between the states.

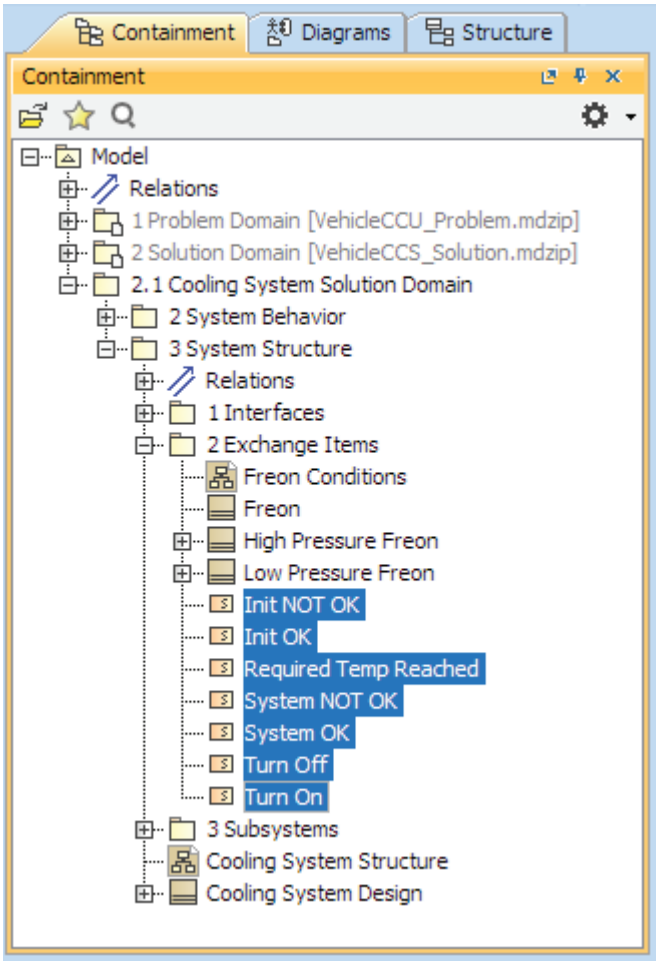


Step 5. Organizing signals into package

You have already created a lot of signals. It's now time to move them into a separate package to keep your model well-organized.

All the signals that trigger the transitions of the Cooling System states are by default placed under the *Cooling System States* state machine. Like all definitions of the items to exchange, these signals should be moved to the 2 Exchange Items

package you have created in step 1 of the SS3 tutorial. Signals can be moved there simply by dragging them onto that package in the Model Browser.



Step 6. Specifying the entry, do, or exit behavior of the state

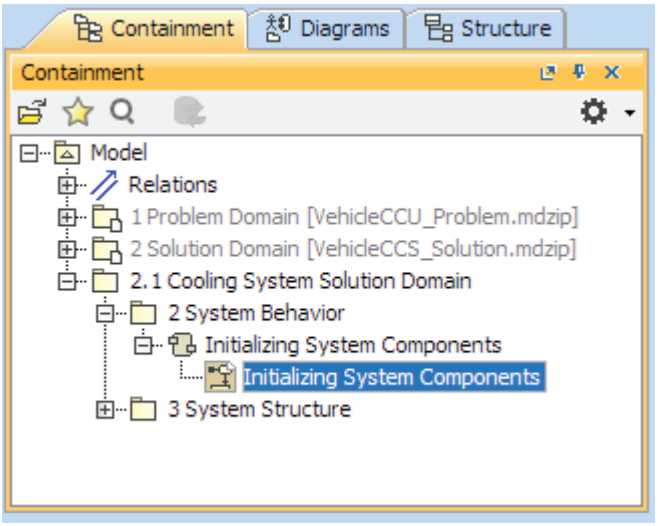
The system or subsystem under design can perform one or more behaviors related to some of its states. Let's say the Cooling System, when in the Initializing state, invokes the Compressor System to start, and waits 60 seconds for the response to the invocation signal. If the response is received, the *Init OK* signal (see step 3 of this tutorial) is sent to the Cooling System. Otherwise, the *Init NOT OK* signal (see step 3 of this tutorial) is sent. This kind of behavior can be captured as SysML activity and stored in the solution domain model of the Cooling System, under the *3 System Behavior* package you have created in step 1 of this tutorial.

There is no need to create the SysML activity and then the diagram. They both appear in the model, after you select to create the SysML activity diagram.

To create a SysML activity diagram

1. Right-click the *2 System Behavior* package and select **Create Diagram**.
2. In the search box, type *ad*, the acronym for the SysML activity diagram, and then press Enter. A SysML activity diagram is created. It is owned by the SysML activity.

3. Type *Initializing System Components* to specify the name of the new diagram and press Enter again. The same name is given to the SysML activity, which owns that diagram.

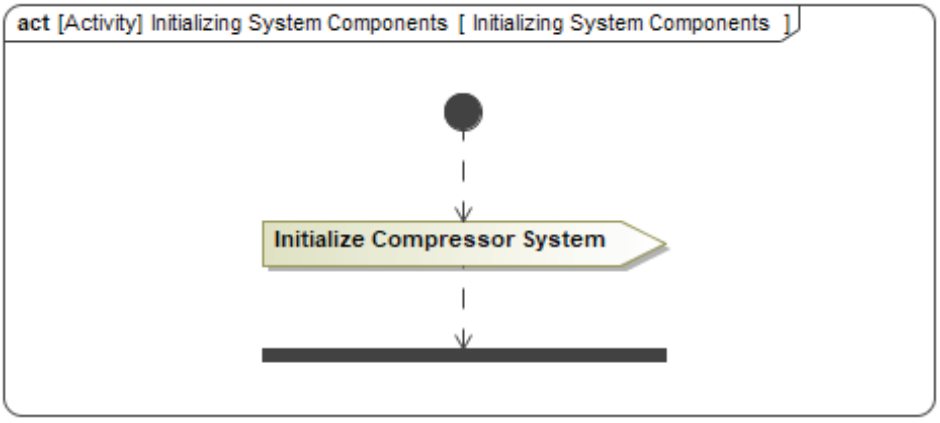


To specify the *Initializing System Components* activity

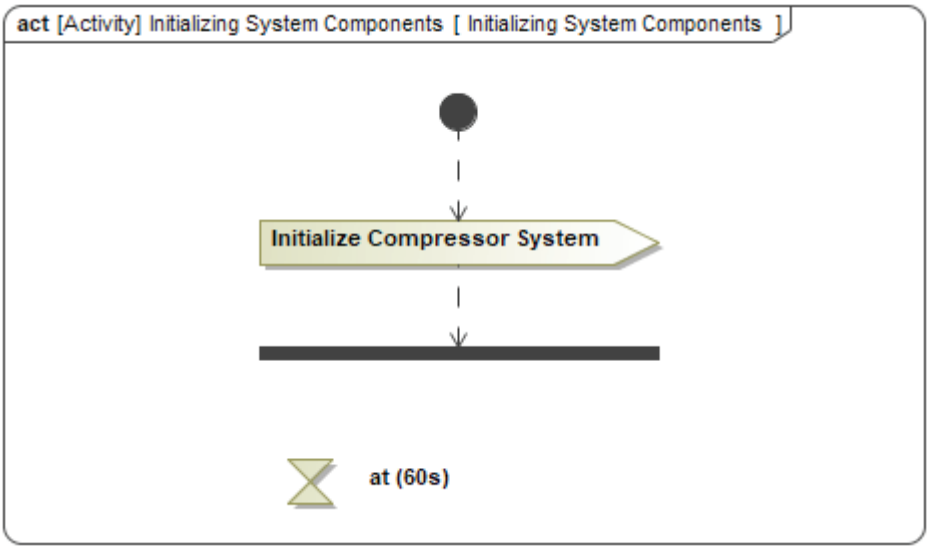
1. Click the **Initial Node** button on the diagram palette and then click an empty place on the diagram pane. The initial node is created and displayed on the diagram.
2. Click the **Send Signal Action** button on the diagram palette and then click an empty place on the diagram pane again. The send signal action is created and displayed on the diagram.
3. Type *Initialize Compressor System* directly on the shape of the created action and press Enter. The new signal with the same name is created in the model, directly under the *3 System Behavior* package.

> ⓘ Recalling what you've learned in step 5 of this tutorial, keep in mind that the new signal should also be moved to the *2 Exchange Items* package. You can do this immediately or as soon as the procedure is finished.

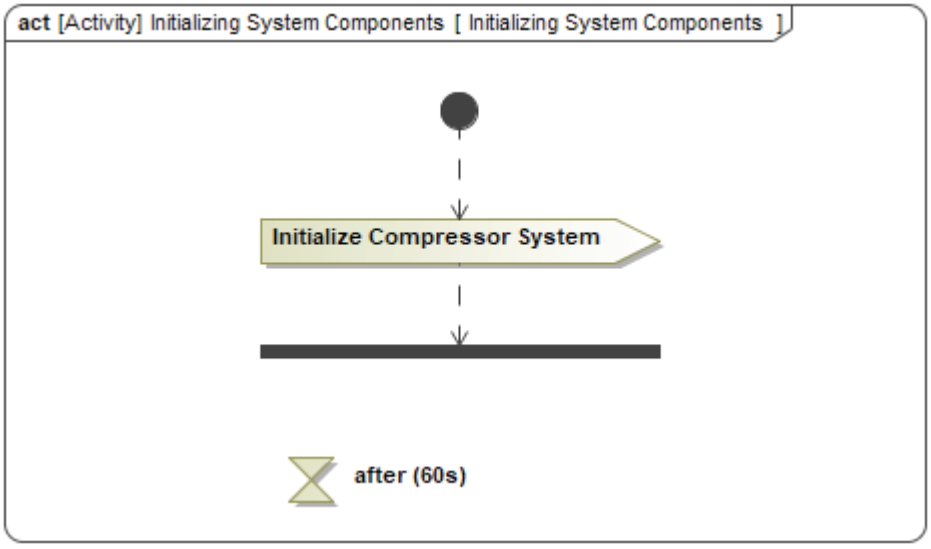
4. Select the shape of the initial node, click the Control Flow button on its smart manipulator toolbar, and then select the shape of the *Initialize Compressor System* send signal action. The control flow is established between the initial node and that send signal action.
5. Select the shape of the *Initialize Compressor System* send signal action, click the Control Flow button on its smart manipulator toolbar, then right-click an empty place on the diagram pane and select **Fork Horizontal**. The horizontal fork is created and displayed on the diagram pane.



6. Click the **Time Event** button on the diagram palette and then click an empty place on the diagram pane again. The time event action is created and displayed on the diagram.
7. Type *60s* directly on the shape of the newly created time event action and press Enter.

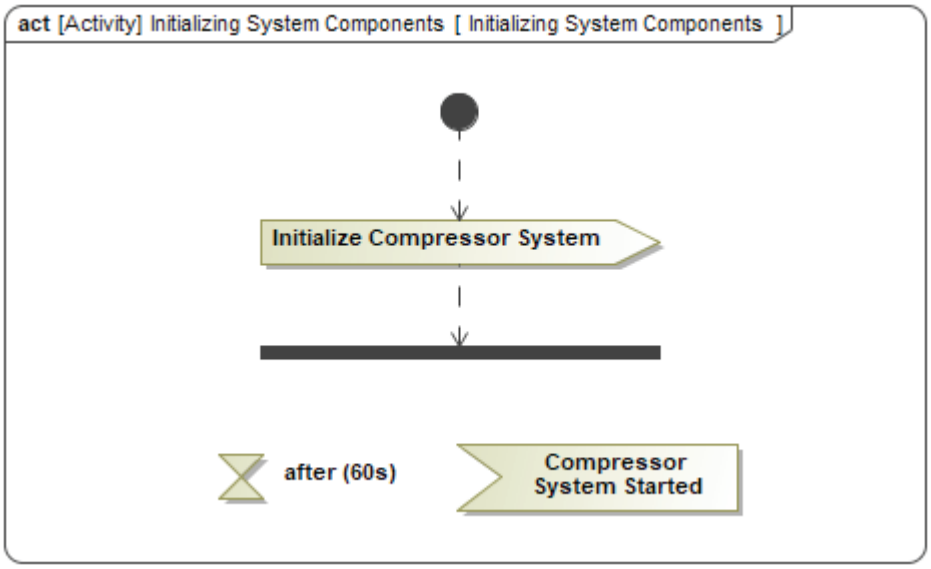


8. Convert the trigger of the time event action from absolute to relative:
   a. Double-click the shape of the time event action to open its Specification.
   b. Find the **Is Relative** property and set its value to *true*.
   c. Close the dialog. The indicator of the trigger changes from *at* to *after*.

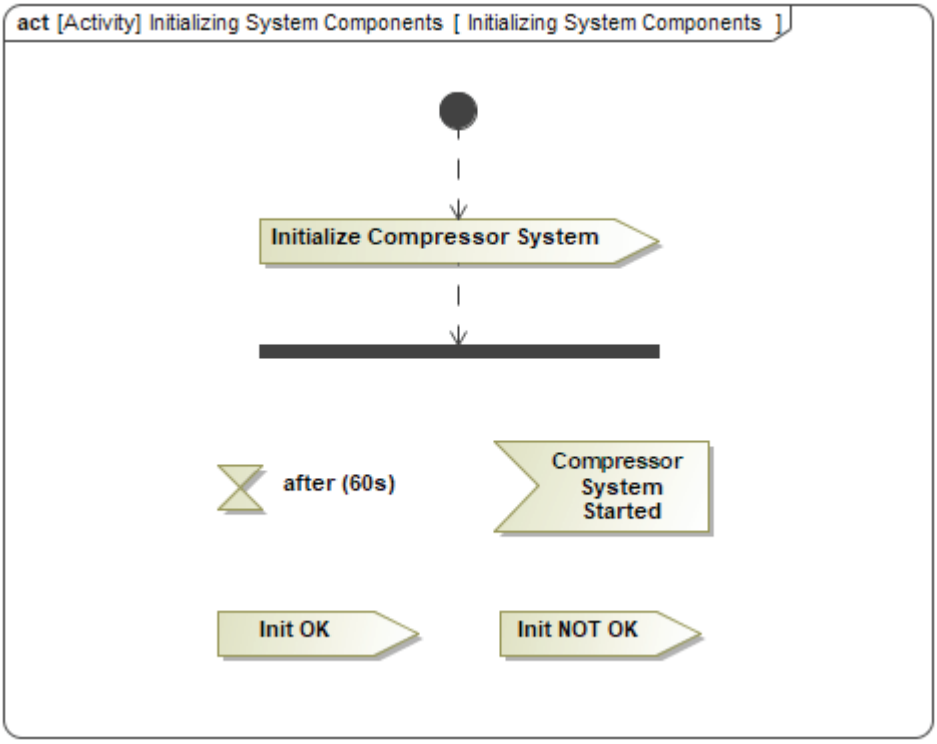


9. Click the **Accept Event Action** button on the diagram palette and then click an empty place on the diagram pane again. The accept event action is created and displayed on the diagram.
10. Create a signal for the newly created accept event action:
    a. In the Model Browser, right-click the *2 Exchange Items* package (see step 4 of this tutorial) and select **Create Element**.
    b. In the search box, type *si*, the first two letters of the element type *Signal*, and press Enter.
    c. Type *Compressor System Started* to specify the name of the new signal and press Enter.

d. Drag the signal to the shape of the accept event action. The action is named after the signal.

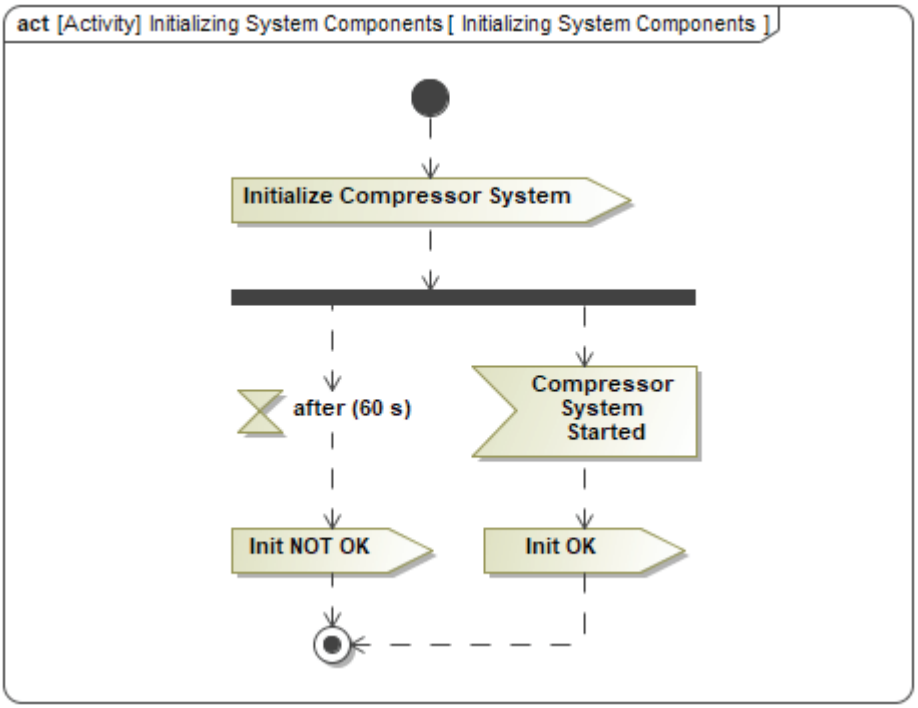


11. Create a send signal action for sending the *Init OK* signal:
    a. In the Model Browser, under the *2 Exchange Items* package (see step 4 of this tutorial), select the *Init OK* signal.
    b. Drag the signal to the diagram pane. The shape of the new action with the signal's name is created on the diagram pane.
12. Create a send signal action for sending the *Init NOT OK* signal the same way as described in step 11.



13. Click the **Activity Final** button on the diagram palette and then click an empty place on the diagram pane. The final node is created and displayed on the diagram.

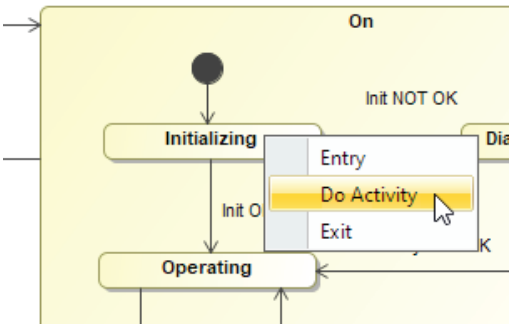
14. Create the rest of the control flows you see in the following figure.



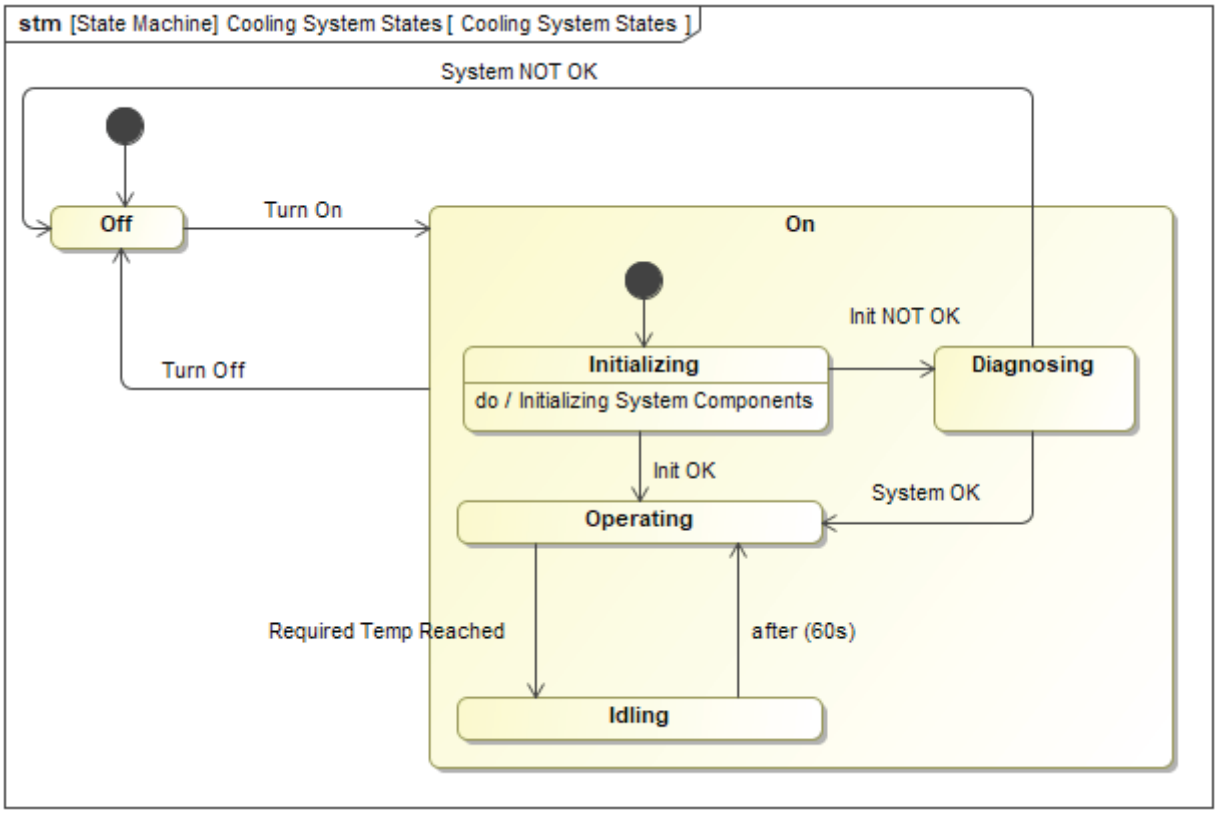
Once the *Initializing System Components* activity is specified, it can be assigned to the *Initializing* state as its *do* behavior.

To set the *Initializing System Components* activity as the do behavior of the *Initializing* state of the Cooling System

1. Open the *Cooling System States* diagram.
2. In the Model Browser, select the *Initializing System Components* activity and drag it to the shape of the *Initializing* state.
3. Click the shape as soon as you see the blue border around the shape.
4. Select **Do Activity** as it is shown in the following figure. The do behavior is set.



The *Initializing* state with the SysML activity assigned as its do behavior is displayed in the following figure. Now you may consider the behavior of the Cooling System specified and move on to integrating the solution domain model of the Cooling System into the solution domain model of the whole system.



S3. System Structure: final

What is it?

After the engineering teams are done with the solution architectures of the subsystems, the systems engineer can integrate them into a single model. As a matter of fact, more than one solution can be proposed; therefore the systems engineer performs a trade-off study to pick a preferred one of each subsystem and build the integrated solution architecture of the whole system.

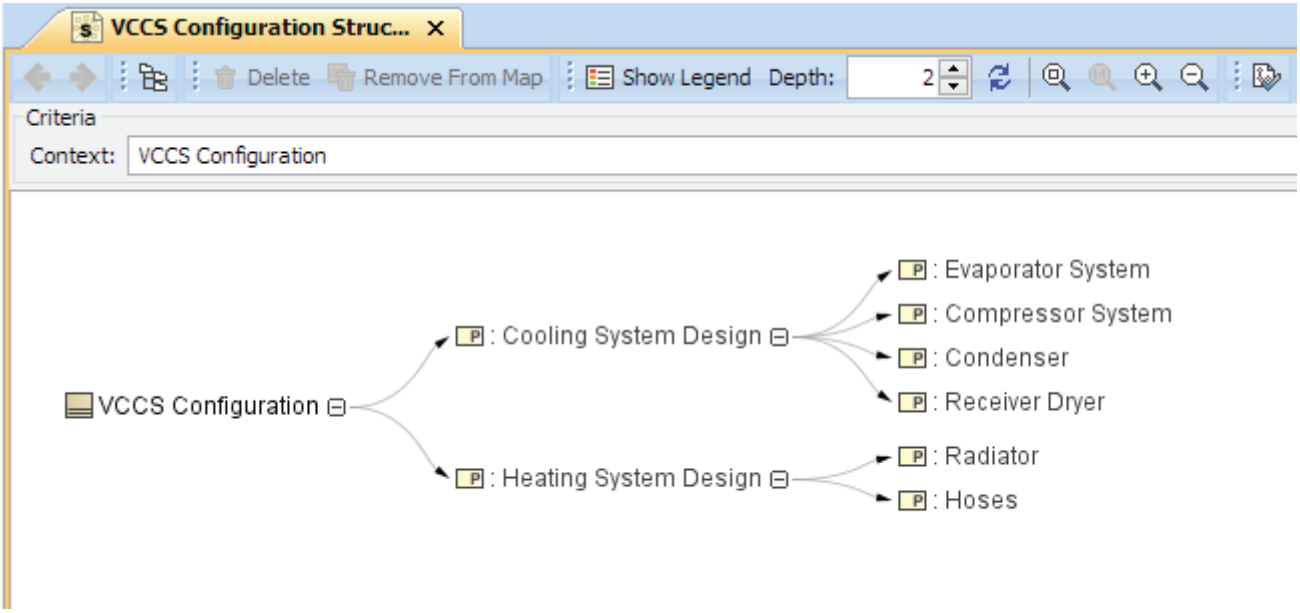
The main purpose of the structure design integration is to verify whether subsystems of the system under design can successfully communicate with each other. Therefore the main task of the systems engineer is to check whether interfaces of separate subsystems are compatible. If the systems engineer detects one or more incompatible interfaces, he/she requests the responsible engineering team to review and update their model appropriately. After the engineering teams complete the request, the systems engineer makes another attempt to integrate the models.

Who is responsible?

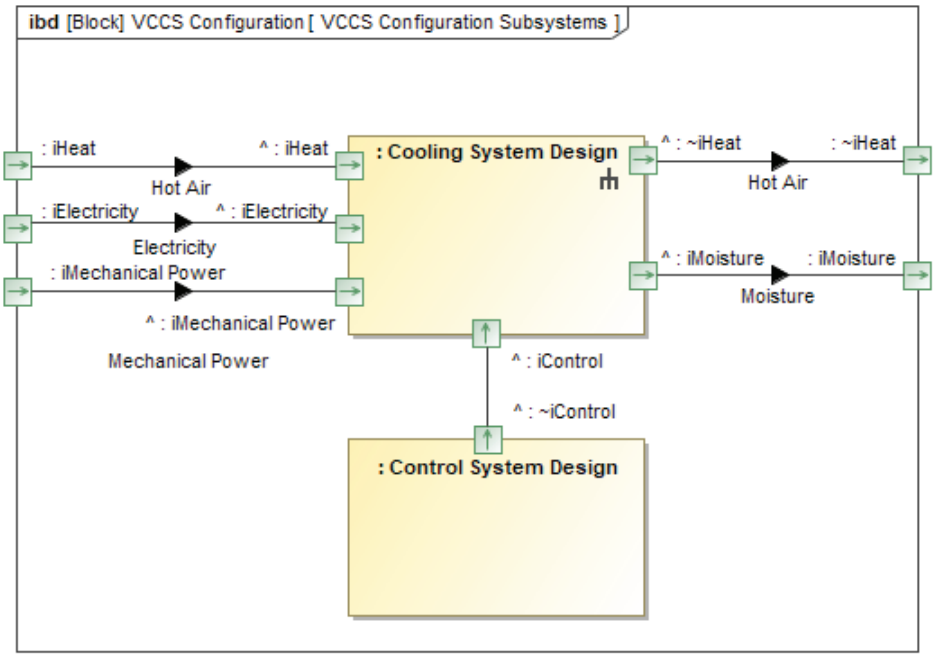
In a typical multidisciplinary systems engineering team, the person responsible for the integration of the subsystems into the whole is the systems engineer, the head of the *Architecture Team*.

How to model?

For the structure integration and review, we recommend utilizing one of the predefined relation maps available in your modeling tool – the Structure Decomposition Map.



For interface compatibility visualization and verification, a SysML internal block diagram can be used.



What's next?

- Having the integrated solution architecture model, you can specify how it satisfies the system requirements. Therefore, you can move to S1.final.
- The integrated solution architecture model enables you to calculate MoEs of the whole system and verify them against the system requirements (see S1.initial). Thus, you can also move to S4.

Tutorial

Step 1. Creating and organizing a model for S3 final

Since you already have a solution architecture of the Cooling System (including the structure and behavior models), you, in the role of the systems engineer, can integrate it into the solution architecture of the whole VCCS (together with the solution architectures of the other subsystems simultaneously developed by the imaginary engineering teams).

For this, you must create a model (another file) that integrates the solution architectures of all the subsystems of the VCCS, including the Cooling System and the Heating System, to name a few. In general, this model can be called the system configuration model. Note that there can be more than one system configuration model within a single solution domain; however, we focus on a single one.

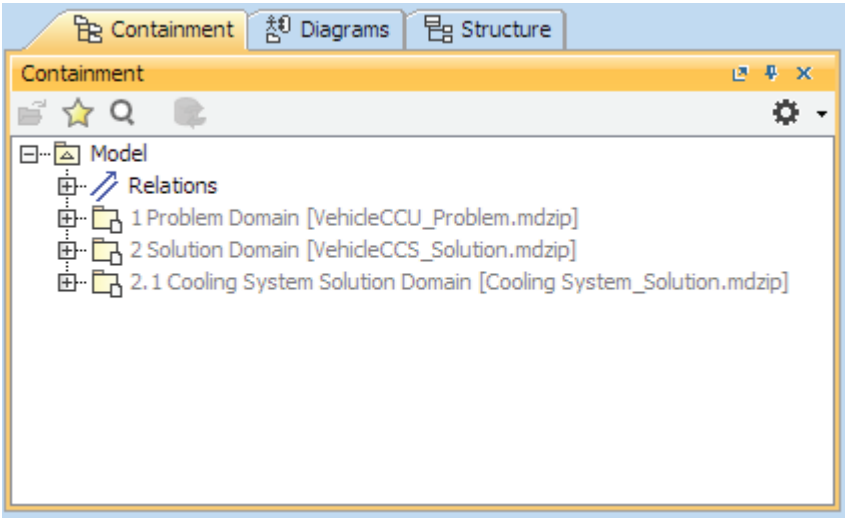
To create a system configuration model of the VCCS

1. Start sharing the solution domain model of the Cooling System.

   ⓘ • In fact, the solution domain models of all subsystems of the VCCS should be shared in this step.
   • For more information on this topic and the topics mentioned in steps 2 and 3, refer to the latest documentation of your modeling tool.

2. Create a new model (file). It can be named *VehicleCCS_Configuration.mdzip*. This is the configuration model of the Vehicle Climate Control System.

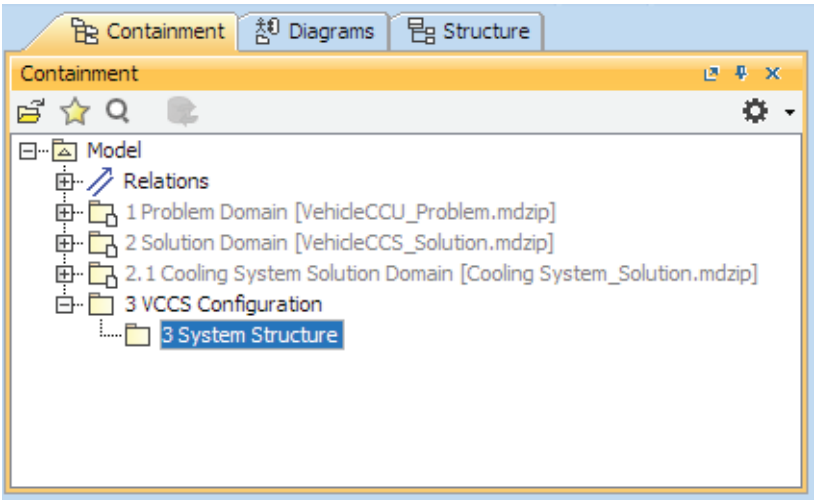
3. Use the solution domain model of the Cooling System as read-only in the configuration model of the VCCS. Having already been used in the solution domain model of the Cooling System, the problem domain model and the solution domain model of the VCCS (*HLSA* model) are automatically used as well.



   ⓘ Grey color element names within the used model indicate that elements cannot be modified. This is because the external model is used in the *read-only* mode.

4. Right-click the *Model* package (this is the default name of the root package) and select **Create Element**.
5. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.
6. Type *3 VCCS Configuration* to specify the name of the new package and press Enter (see the figure in step 9).
7. Right-click the *3 VCCS Configuration* package and select **Create Element**.
8. Repeat step 5.

9. Type *3 System Structure* to specify the name of the new package and press Enter.



Step 2. Capturing the integrated structure of the system configuration

For the structure integration and review, we recommend utilizing one of the predefined relation maps available in your modeling tool – the structure decomposition map. It can be created in the *3 System Structure* package within the VCCS configuration model you created in the previous step.

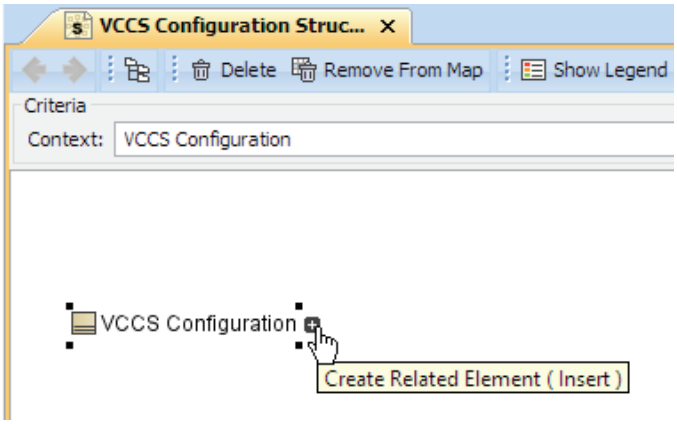
To capture the integrated structure of the VCCS configuration with the structure of the Cooling System

1. Create a block to capture the VCCS configuration:
   a. In the Model Browser, right-click the *3 System Structure* package you created in step 1 of this tutorial and select **Create Element**.
   b. In the search box, type *bl*, the first two letters of the element type *Block*, and press Enter.
   c. Type *VCCS Configuration* to specify the name of the new block and press Enter.
2. Create the structure decomposition map:
   a. Right-click the VCCS Configuration block and select **Create Diagram**.
   b. In the search box, type sdm, where s stands for structure, d for decomposition, and m for map, and then press Enter. The diagram is created, and the VCCS Configuration block is set as its context.

      ⓘ If you don't see any result, click the **Expert** button below the search results list. The list of available diagrams expands in the Expert mode.

   c. Type *VCCS Configuration Structure* to specify the name of the new diagram and press Enter again.
3. Create a part property for the *VCCS Configuration* block to integrate the structure model of the Cooling System into the one of the whole system configuration:

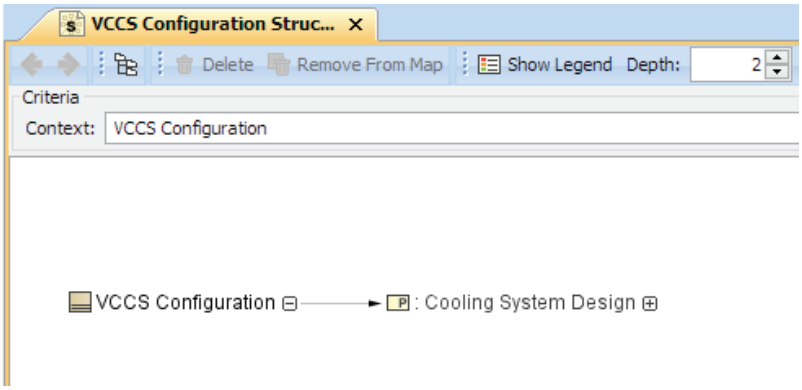
a. Select the *VCCS Configuration* block on the diagram pane and click the Create Related Element button (see the following figure). An unnamed part property is created.



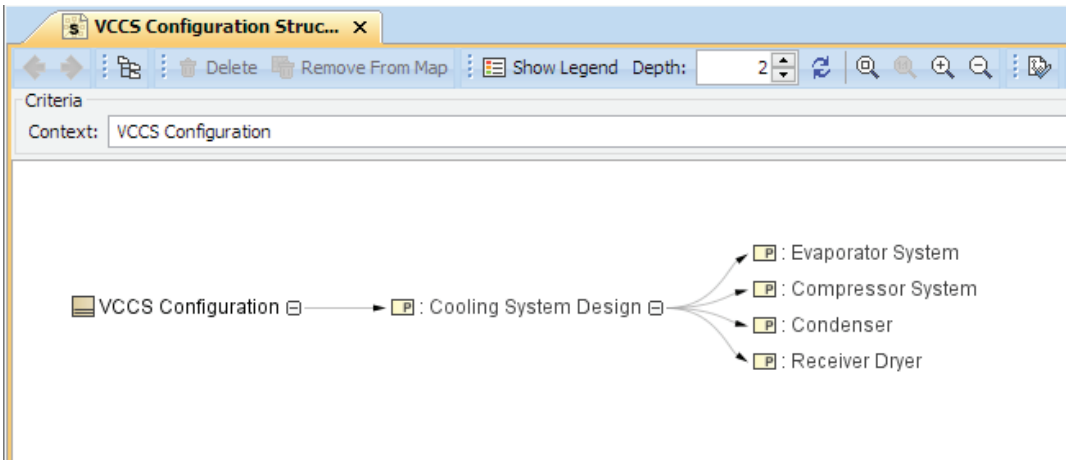
b. Type *:Cool*, press Ctrl + Spacebar, and then select the *Cooling System Design* block.

> ⓘ If you don't see that block on the list, click to clear the *Apply Filter* check box at the bottom-left corner of the dialog. The filter by default excludes elements that are stored in external models, and the *Cooling System Design* block is apparently one of those – it comes from the solution domain model of Cooling System.

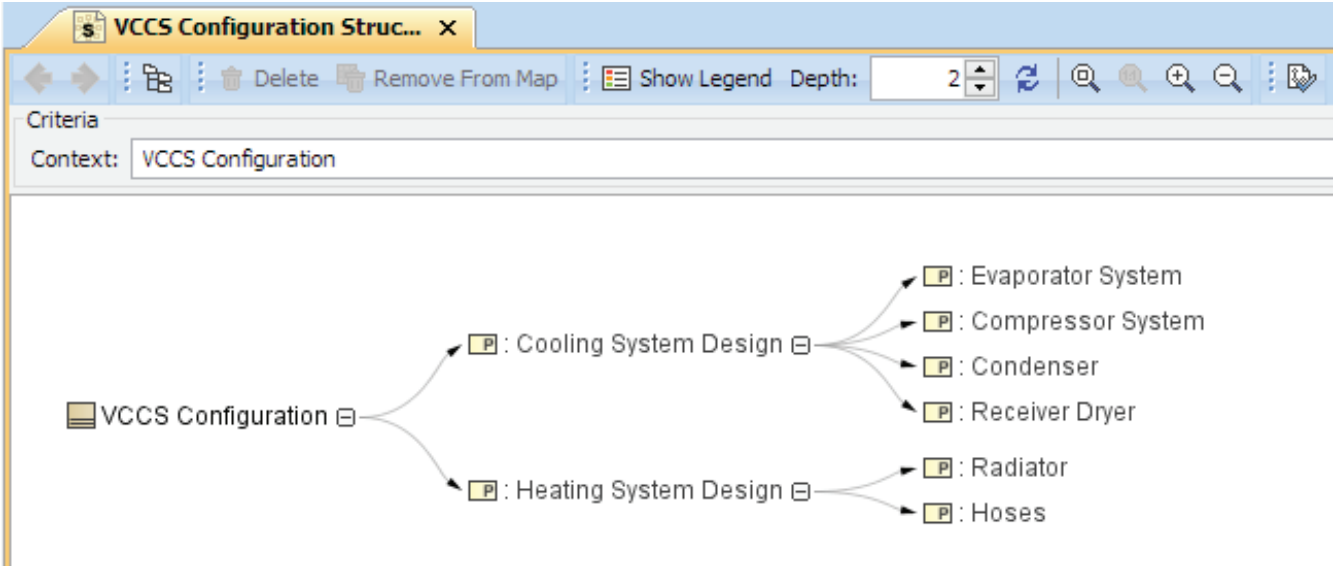
c. Press Enter to finish. The new part property is created for the *VCCS Configuration* block (see the following figure). It is typed by the *Cooling System Design* block, which means that the structure of the Cooling System designed by the separate engineering team in a separate model is now integrated into the structure model of the entire VCCS.



d. Click the Suppress/Expand button on the shape of the part property typed by the *Cooling System Design* block to display the deeper structure of the Cooling System.



When the engineering team responsible for another subsystem design provides the structure of that subsystem to the systems engineer, he/she integrates it into the whole in the same way as described above. The following figure displays a more elaborate structure model of the VCCS with the integrated structure of the Heating System.

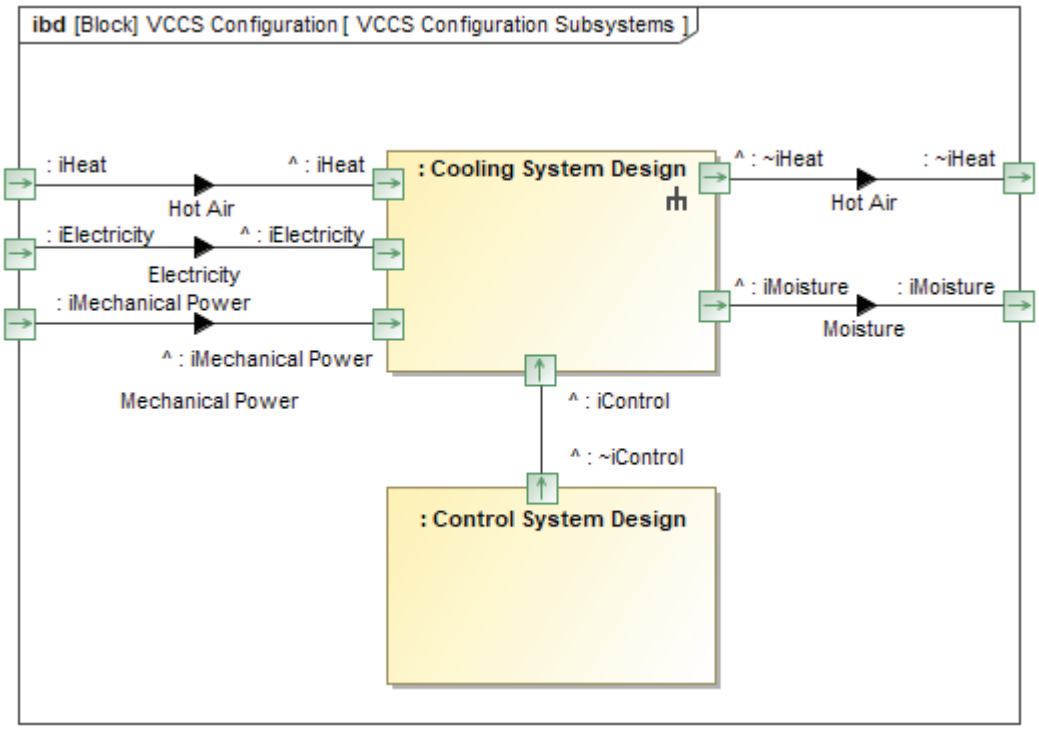


Step 3. Verifying interface compatibility between subsystems

The main purpose of the structure design integration is to verify whether subsystems of the system under design can successfully communicate with each other as well as with the outside of the system. Therefore the main task of the systems engineer is to check whether interfaces of separate subsystems are compatible.

For interface compatibility visualization and verification, a SysML internal block diagram can be used. As you can see in the following figure (we assume that you created it on your own), the Cooling System can successfully communicate with the Control System, another subsystem of the VCCS. In terms of SysML, the communication is established via the proxy ports typed by the iControl interface. To check whether the Cooling System can successfully communicate with the outside of the VCCS (this is illustrated by the connectors to the proxy ports on the diagram frame), you need to see the bigger picture; the picture of the entire vehicle system. So far, you can only assume that the whole system of the vehicle has an engine, which provides the mechanical power to the Cooling System, and the alternator, which provides the electricity, and so on.

Also, please note that the diagram should be updated after the integration of the rest of the subsystems of the VCCS.



S4. System Parameters

What is it?

System parameters can be specified as soon as the system structure is captured in the model. System parameters capture quantitative characteristics of the system under design. They satisfy non-functional system requirements (see S1.initial) and are based on measurements of effectiveness (see B4 and/or W4).

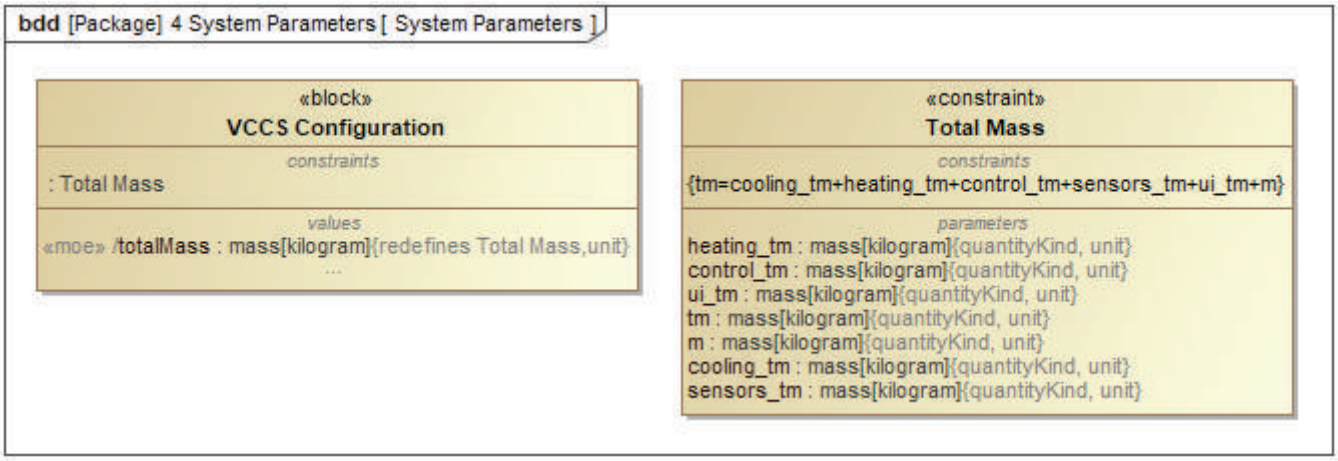
System parameters are derived from other values; for example, subsystem parameters. Mathematical expressions, which define the rules of derivation, can be specified in the model. Knowing how to calculate system parameter values enables early system requirements verification. Using the capabilities of the modeling tool, the model can be executed to calculate these values and give the verdict on whether quantitative requirements are satisfied or not.

Who is responsible?

In a typical multidisciplinary systems engineering team, system parameters and formulas for calculating them are captured in the model by the *System Analysis Team*.

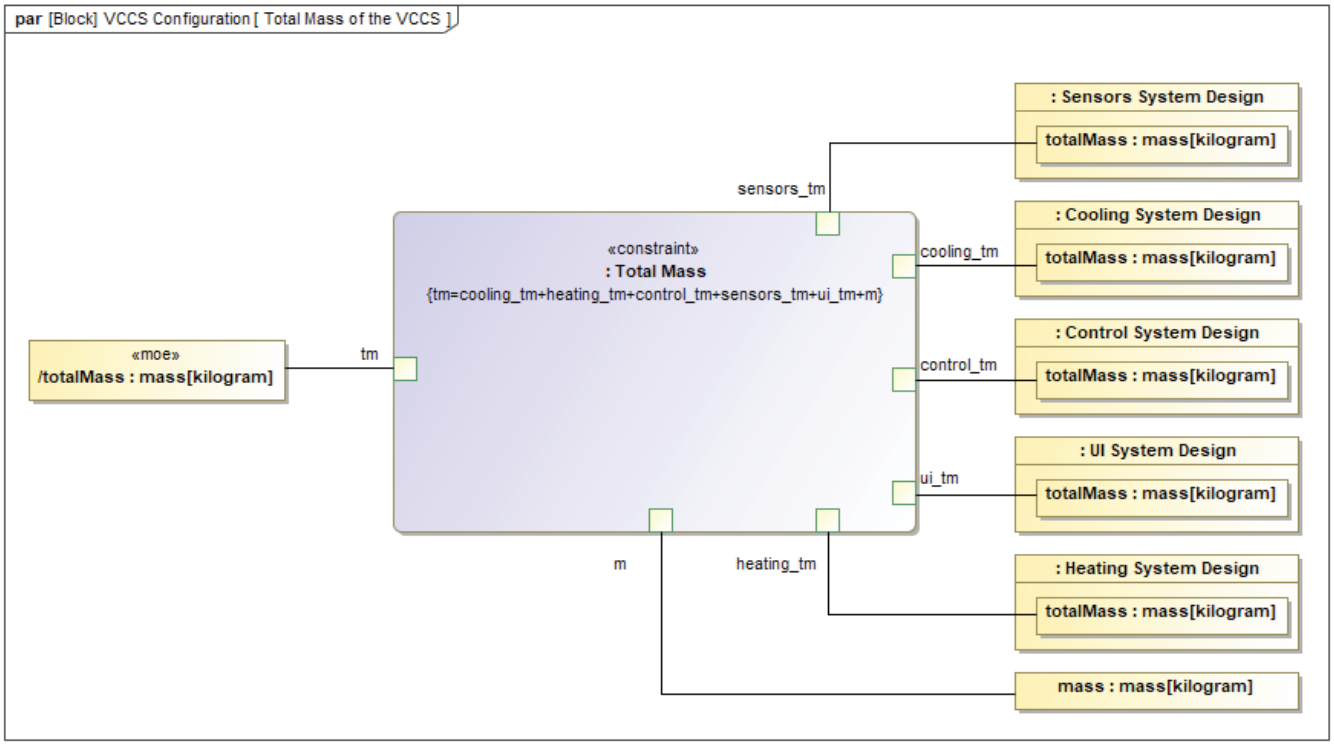
How to model?

In the modeling tool, a system parameter can be captured as a value property of the block, which may represent a system, subsystem, or component. The formula for calculating that system parameter can be captured as a constraint expression of a constraint block. The infrastructure of the SysML block definition diagram can be utilized for capturing both the system parameter and the constraint expression. The following figure displays the diagram that captures a system parameter as the *totalMass* value property and a formula for calculating this parameter as the constraint expression of the *Total Mass* constraint block.

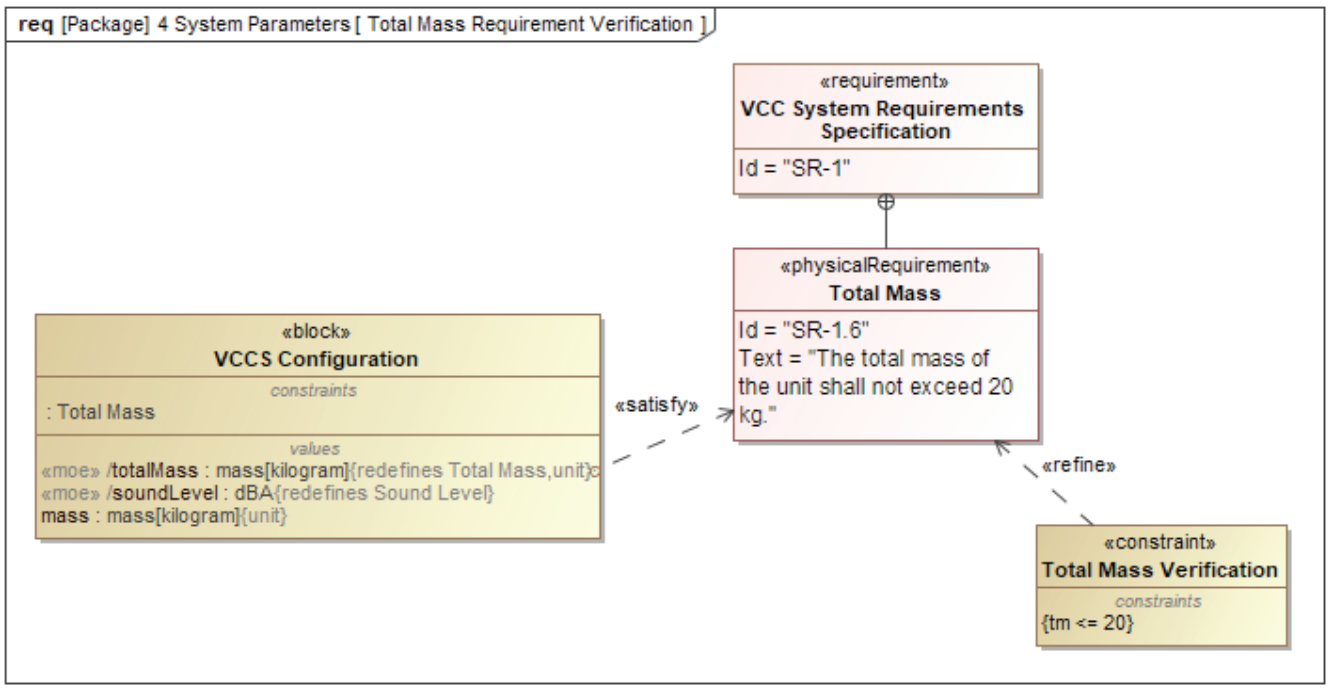


To empower the constraint expression to perform calculations, you have to specify what system, subsystem, or component parameters captured somewhere in your model (or even across multiple models of the solution domain) should be

consumed as variables (or constraint parameters) of that constraint expression. This can be done in the SysML parametrics diagram by utilizing binding connectors represented as solid lines in the following figure.



The simulation capabilities of the modeling tool allows you to calculate system parameters with given inputs. Having the system parameter calculated, you can verify the appropriate non-functional system requirement and give the verdict on whether it is satisfied or not. The modeling tool enables you to perform this verification automatically. To get ready for the automated requirements verification, you need to relate the value property, which captures the system parameter, to the appropriate system requirement. You need to specify a satisfy relationship between them. The following figure displays an example of the satisfy relationship between the *totalMass* value property and the *Total Mass* nonfunctional requirement, which means that the requirement is satisfied when the value of the *totalMass* value property is not greater than 20 kg. As you can see, the natural language expression in the requirement text is refined by another constraint expression of the *Total Mass* constraint block, the *tm <= 20* inequality.



As you can see, both satisfy and refine relationships are represented in the SysML requirements diagram, but the bdd can also be used for this purpose.

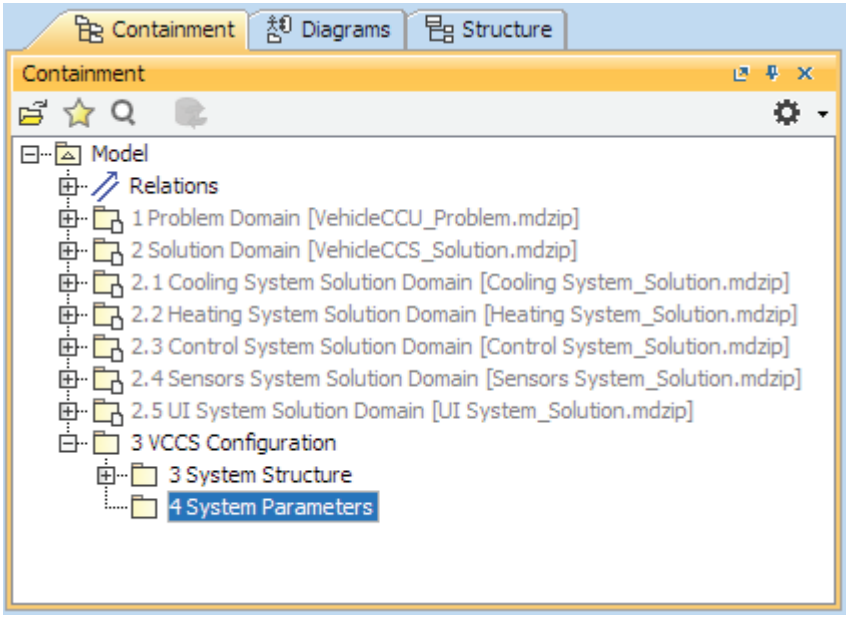
What's next?

- Once you have system parameters, you can specify how they satisfy system requirements. Therefore, you can move to S1.final.

Tutorial

Step 1. Organizing the model for S4

Let's assume that you want to specify how to calculate the total mass of the Vehicle Climate Control System. Since that system parameter is related to the whole system instead of being tied to some subsystem or component, it should be specified in the system configuration model you updated in step 3 of the S3 final phase tutorial. This and all subsequently captured system parameters can be stored inside the *4 System Parameters* package placed directly under the *3 VCCS Configuration* package in the system configuration model.

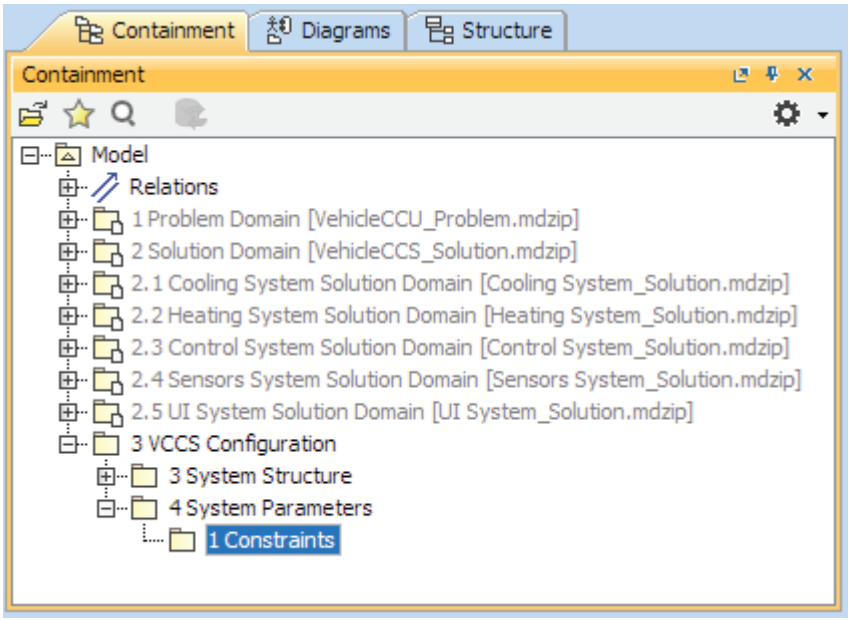


ⓘ Calculating the total mass of the Vehicle Climate Control System requires knowing the total mass of each subsystem. These are subsystem parameters that come from solution domain models of subsystems. In order to use these parameters for calculating the total mass of the VCCS, solution domain models of each subsystem must be used in the system configuration model. This is why the preceding figure displays them as used in the system configuration model. However, you should keep in mind that the solution domain models of subsystems, except for the one of the Cooling System, are not elaborated. They are needed only for capturing appropriate subsystem parameters. You learn how to create subsystem parameters in one of the subsequent steps of this cell tutorial.

To organize the model for S4

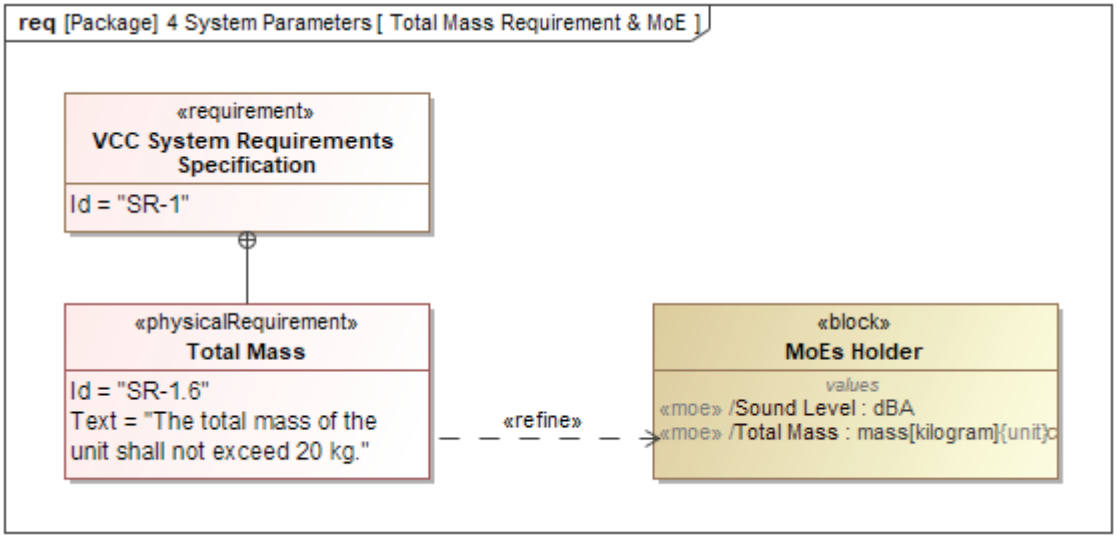
1. Open the system configuration model of the VCCS (the *VCCS_Solution.mdzip* file) you created in step 1 of the S3 final phase tutorial, if not opened yet.
2. Right-click the *3 VCCS Configuration* package and select **Create Element**.
3. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.
4. Type *4 System Parameters* to specify the name of the new package and press Enter.

You also need to create a package for storing constraint blocks in the model. Let it be the *1 Constraints* package under the *4 System Parameters* package you see in the following figure (we assume that you created it on your own).



Step 2. Specifying a system parameter to capture the total mass

As you can see in the following figure, system requirements specification (see S1) and measurements of effectiveness (see B4) determine the specification of the system parameter that captures the total mass of the VCCS.



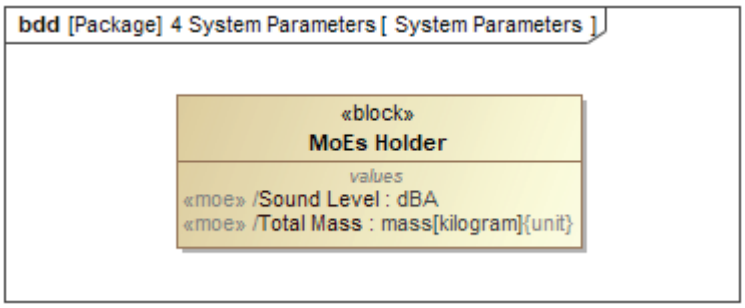
System parameters are captured as value properties of a relevant block. In this case, you should create a value property for the VCCS block in the system configuration model. However, using the inheritance and redefinition capabilities of the modeling tool makes the start easier. Let's see how it works.

To specify a system parameter that captures the total mass of the VCCS

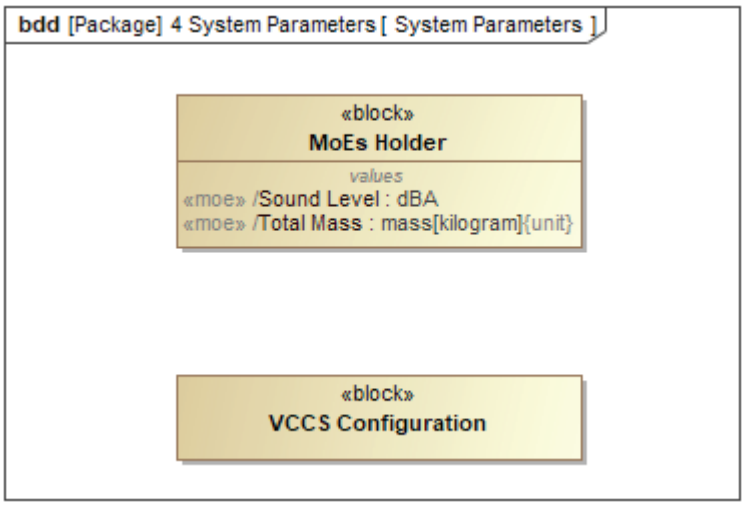
1. Create a bdd for capturing system parameters:
   a. Right-click the *4 System Parameters* package you created in step 1 of the S4 tutorial and select **Create Diagram**.
   b. In the search box, type *bdd*, the acronym of SysML block definition diagram, and then double-press Enter. The diagram is created.

   ⓘ The diagram is named after the package where it is stored. This name perfectly works for the diagram, too. You may only remove the sequence number from its name.

2. Display the *MoEs Holder* block on the diagram:
   a. Press Ctrl + Alt + F to open the **Quick Find** dialog.
   b. In the dialog:
      iii. Click **Any Element** to search for particular elements only.
      iv. Type *moes*.
      v. Click to clear the **Apply Filter** check box at the bottom of the search results list to include the contents of external models into the search scope (this is necessary as the *MoEs Holder* block resides in the problem domain model).
   c. When you see the *MoEs Holder* block selected in the search results list, press Enter. The block is selected in the Model Browser.
   d. Drag the *MoEs Holder* block to the diagram pane. The shape of the block is displayed on the diagram.



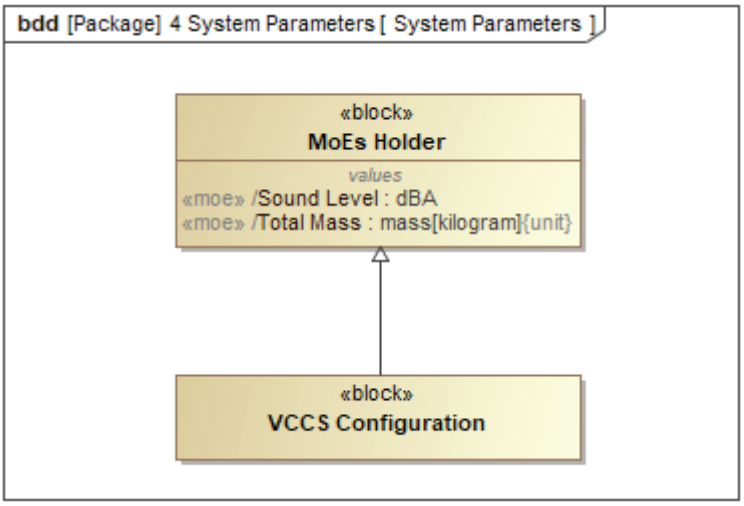
3. Display the *VCCS Configuration* block on the diagram:
   a. Press Ctrl + Alt + F to open the **Quick Find** dialog.
   b. In the dialog:
      i. Click **Any Element** to search for particular elements only.
      ii. Type *vccs*.
   c. When you see the *VCCS Configuration* block selected in the search results list, press Enter. The block is selected in the Model Browser.
   d. Drag the *VCCS Configuration* block to the diagram pane. The shape of the block is displayed on the diagram.



4. Establish the generalization between the *VCCS Configuration* block (sub-type) and the *MoEs Holder* block (super-type):
   e. Click the Generalization button on the diagram palette.

f. Click the shape of the *VCCS Configuration* block.

g. Click the shape of the *MoEs Holder* block. The *MoEs Holder* block becomes the super-type of the *VCCS Configuration* block, which means that the latter inherits all value properties from the former.



5. Show inherited value properties on the shape of the *VCCS Configuration* block:

   a. Right-click the shape and select **Symbol Properties**.

   b. Click the drop-down list box above the properties list and select **All**. All properties of the block shape are displayed in the dialog.

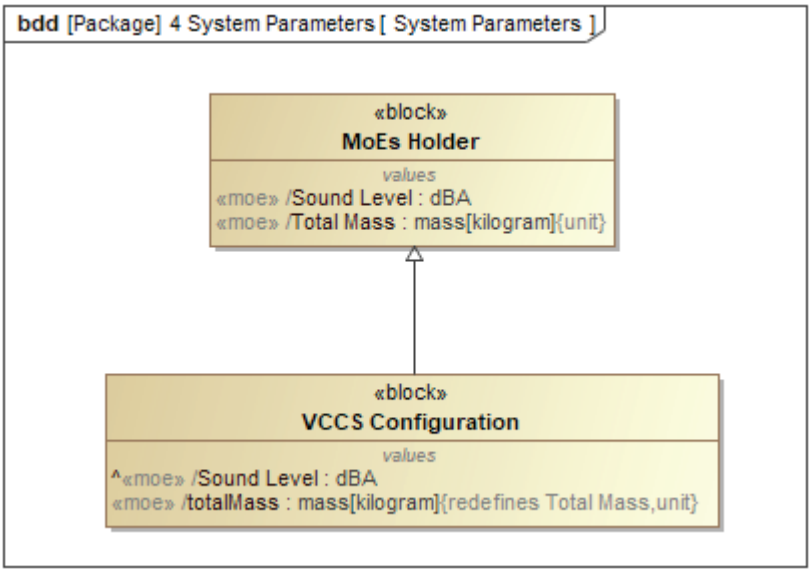
   c. In the search field at the bottom of the open dialog, type *inh*. The **Show Inherited** property appears at the top of the property list.

   d. Click to set the property value to *true* and close the dialog. The inherited value properties of *VCCS Configuration* block are displayed on its shape. Caret symbols ("**^**") indicate them as inherited.

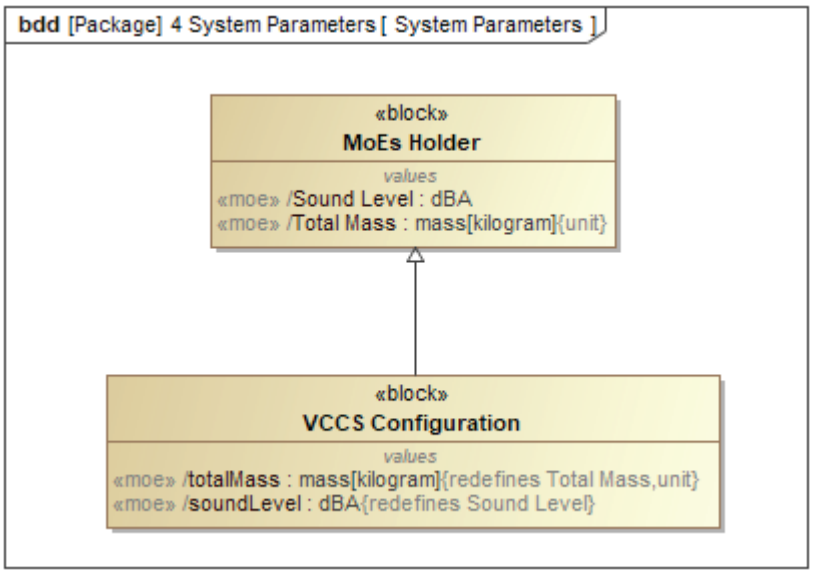
6. Redefine the *Total Mass MoE* for the *VCCS Configuration* block:

   a. Right-click the *Total Mass* value property on the shape of that block and select **Refactor** > **Redefine To**.

   b. Rename the value property to *totalMass* directly on the shape and press Enter. Let this be a new convention of naming value properties in the solution domain.



You may also redefine the *Sound Level MoE*, though we're not going to specify how to calculate it. Afterwards, your *System Parameters* diagram should look like the one in the following diagram.



Step 3. Capturing a formula for calculating the total mass

The *totalMass* value property captures a derivative system parameter. Therefore, the next thing you need to do is capture the mathematical expression which specifies how to calculate that value property. The mathematical expression can be captured as the constraint expression of some constraint block stored in your model.

In this step, you should define a constraint block and specify a constraint expression for calculating the total mass of the VCCS. Let's say the total mass of the VCCS is a sum of masses of all its subsystems plus the mass of the VCCS itself. The constraint expression can be defined as follows:

tm = cooling_tm + heating_tm + control_tm + sensors_tm + ui_tm + m

where:

1. *tm* stands for the total mass of the VCCS
2. *cooling_tm* stands for the total mass of the Cooling System
3. *heating_tm* stands for the total mass of the Heating System
4. *control_tm* stands for the total mass of the Control System
5. *sensors_tm* stands for the total mass of the Sensors System
6. *ui_tm* stands for the total mass of the Sensors System
7. *m* stands for the mass of the VCCS itself

In terms of SysML, t*m, cooling_tm, ...,* and *m* are generally referred to as constraint parameters and later should be bound to corresponding value properties specified in the model.

A bdd can be utilized for capturing constraint blocks. This constraint block is not an exception; therefore, you can use the bdd you created in step 2 of this tutorial. The constraint block can then be moved to the *1 Constraints* package you created in step 1 of this tutorial.

ⓘ Usually constraint blocks are first defined in the model and only then are applied to blocks. In terms of SysML, a constraint block is applied to some block when it types a constraint property of that block. A single constraint block, defined in the model once, can then be applied to many blocks. Therefore, constraint blocks should be stored in a separate package within the model.

To capture the formula for calculating the total mass of the VCCS

1. Create a constraint block:

   b. Open the *System Parameters* diagram, if not opened yet.

   c. Click the **Constraint Block** button on the diagram palette and then click the empty place of that diagram pane. An unnamed constraint block is created.

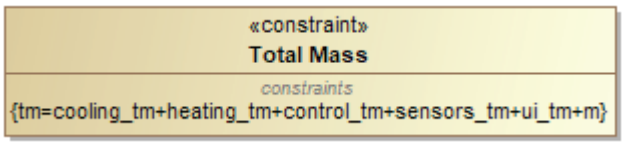
   d. Type *Total Mass* directly on the shape of that element to specify its name and press Enter.

2. Specify a constraint expression to capture the above defined formula:

   a. Click the symbol of the empty constraint expression on the *Total Mass* constraint block.

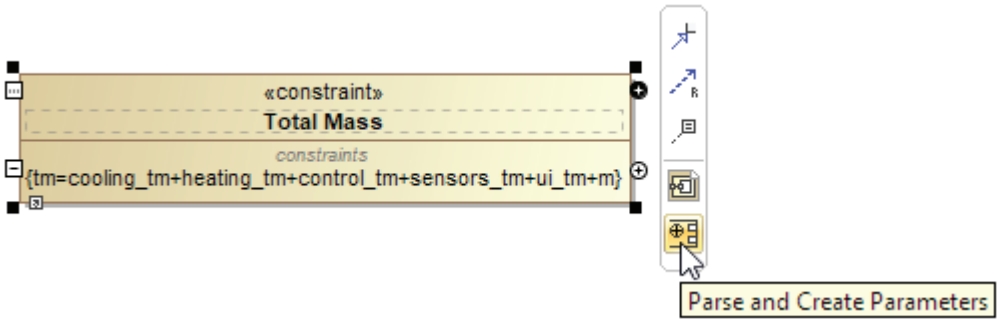
   b. Click the selection again. The empty constraint expression switches to the edit mode.

   c. Type t*m=cooling_tm+heating_tm+control_tm+sensors_tm+ui_tm+m* and then press Enter.



3. Specify constraint parameters:

   a. Select the shape of the *Total Mass* constraint block and click the Parse and Create Parameters button on its smart manipulator toolbar (see the following figure). Constraint parameters are automatically extracted from the constraint expression and displayed in the parameters compartment box on the shape.



   b. Select parameters one by one and change their types from the default *Real* to *mass[kilograms]* directly on the shape of the constraint block:

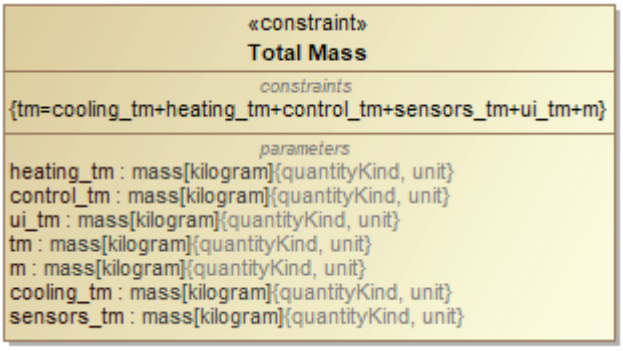
      i. Double-click the type to select it, type *mass[k*, and press Ctrl + Spacebar.

      ii. Select the *mass[kilograms]* type from the list below.

> ⓘ If the list is empty, you need to load the ISO library. For this, click **ISO** on the smart manipulator toolbar on the selected parameter.

iii. Press Enter.

Once you're finished with the last one, the shape of the *Total Mass* constraint block should look like the one in the following figure.

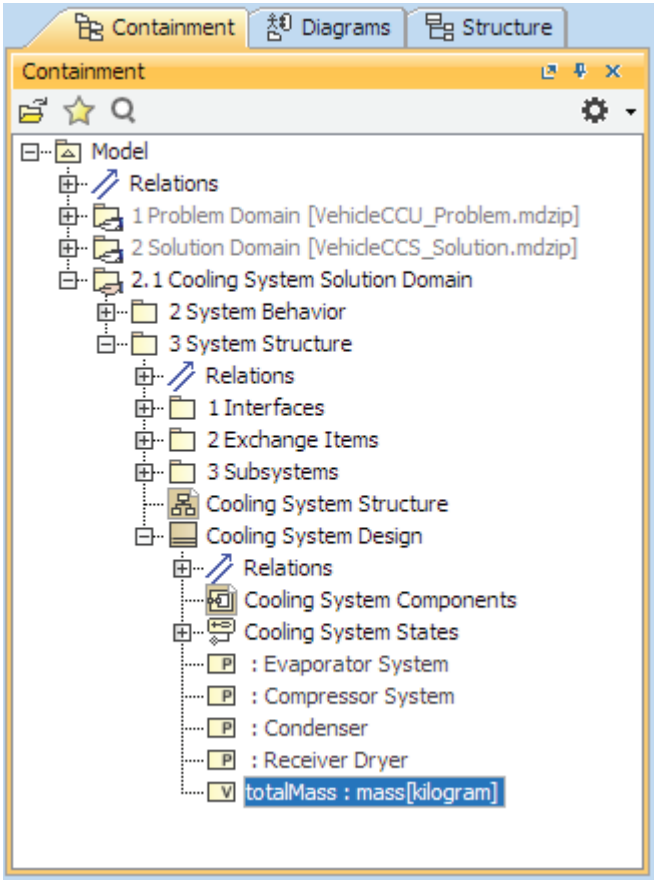


4. Drag the *Total Mass* constraint block to the *1 Constraints* package in your model. The constraint block is moved to that package.

Step 4. Specifying subsystem parameters to capture total masses

Having the constraint expression, you can easily determine what system/subsystem parameters are necessary for calculating the total mass of the VCCS. Input parameters of that constraint expression can help you with this task.

As you may recall, the constraint expression of the *Total Mass* constraint block has six input parameters and therefore requires six system/subsystem parameters captured as value properties in your models. Five of them are subsystem parameters, each capturing the total mass of one subsystem. They should be specified in separate models, each in the solution domain model of the relevant subsystem. For example, the value property capturing the total mass of the Cooling System should be defined in the solution domain model of the Cooling System you created in step 1 of the SS3 tutorial. As you can see in the following figure, the totalMass value property should be created for the *Cooling System Design* block.



To specify a subsystem parameter that captures the total mass of the Cooling System

1. Open the solution domain model of the Cooling System, the one you created in step 1 of the SS3 tutorial.
2. In the Model Browser, select the *Cooling System Design* block:
   a. Press Ctrl + Alt + F to open the **Quick Find** dialog.
   b. In the dialog:
      i. Click **Type** to search for typing elements only.
      ii. Type *cool*.
   c. When you see the *Cooling System Design* block selected in the search results list, press Enter. The block is selected in the Model Browser.
3. Specify the *totalMass* value property for the *Cooling System Design* block:
   a. Right-click the *Cooling System Design* block and select **Create Element**.
   b. In the search box, type *vp*, where *v* stands for *value* and *p* for *property*, and press Enter.
   c. Type *totalMass:mass[k* and press Ctrl + Spacebar to select the suitable value type.
   d. Select the *mass[kilograms]* value type from the list below and press Enter.

The total masses of the other subsystems can be captured appropriately, following the above described steps.

The last of the six input parameters of the constraint expression requires you to specify another system parameter – the mass of the VCCS itself. The appropriate value property can be captured in the next step.

Step 5. Binding constraint parameters to system/subsystem parameters

Once you have the system/subsystem parameters (captured as value properties in the model), it's time to bind them to the appropriate constraint parameters of the *Total Mass* constraint block. Remember that it is the binding which empowers the constraint expression to perform calculations on given inputs.

Bindings between value properties and constraint parameters can be established using binding connectors. This type of SysML relationship can be created by utilizing the infrastructure of the SysML parametrics diagram created for the *VCCS Configuration* block. However, creating the SysML parametrics diagram is not enough to get started. Establishing binding connectors is possible only after the Total Mass constraint block is applied to the *VCCS Configuration* block.

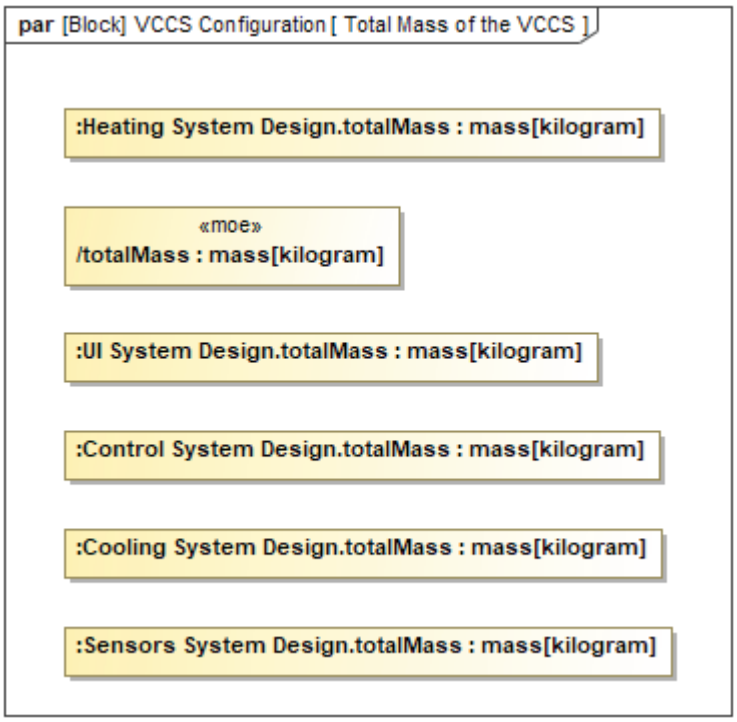
To bind the constraint parameters of the *Total Mass* constraint block to the corresponding value properties

1. Open the VCCS configuration model and be sure the used models are up to date.

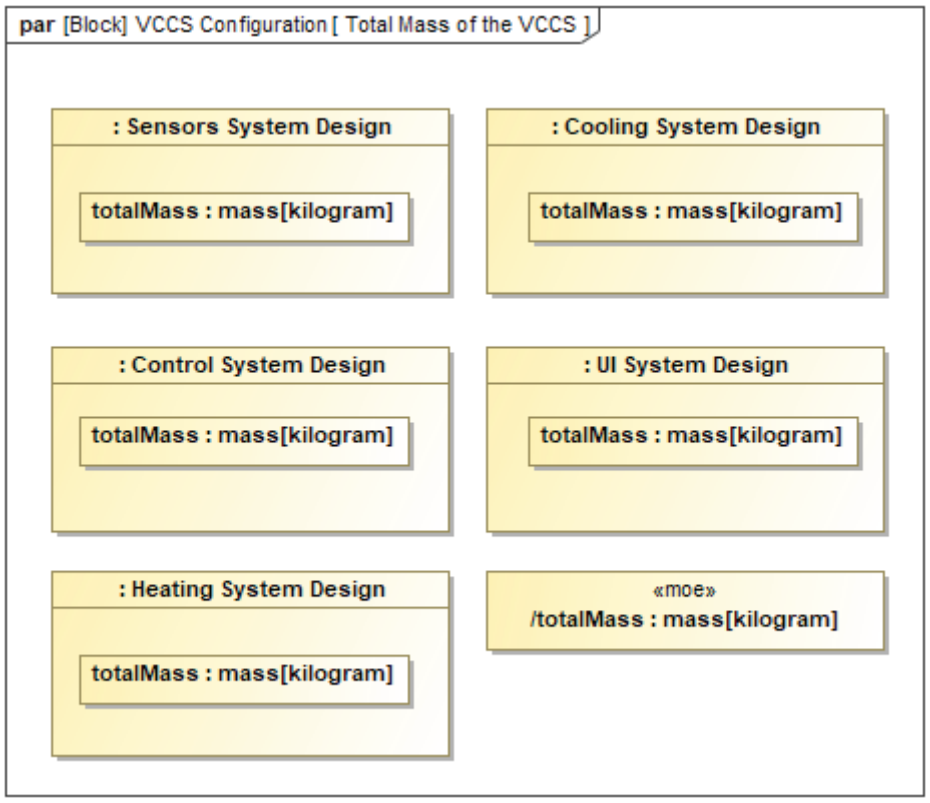
> ⓘ For more information on this topic, refer to latest documentation of your modeling tool.

2. Find the *VCCS Configuration* block:
   a. Press Ctrl + Alt + F. The **Quick Find** dialog opens.
   b. Type *vccs*.
   c. When you see the *VCCS Configuration* block selected in the search results list, press Enter. The block is selected in the Model Browser.
3. Create a SysML parametrics diagram for the *VCCSystem Configuration* block:
   a. Press Ctrl + N. The **Create Diagram** dialog opens.
   b. In the search box, type *par*, the acronym for the SysML parametrics diagram, and press Enter. The blank SysML parametrics diagram is created, and the **Display Parameters/ Parts** dialog opens.

c. Select all the *totalMass* value properties and click **OK**. Selected value properties are displayed on the diagram pane, and the diagram is selected in the Model Browser with the name edit mode switched on.



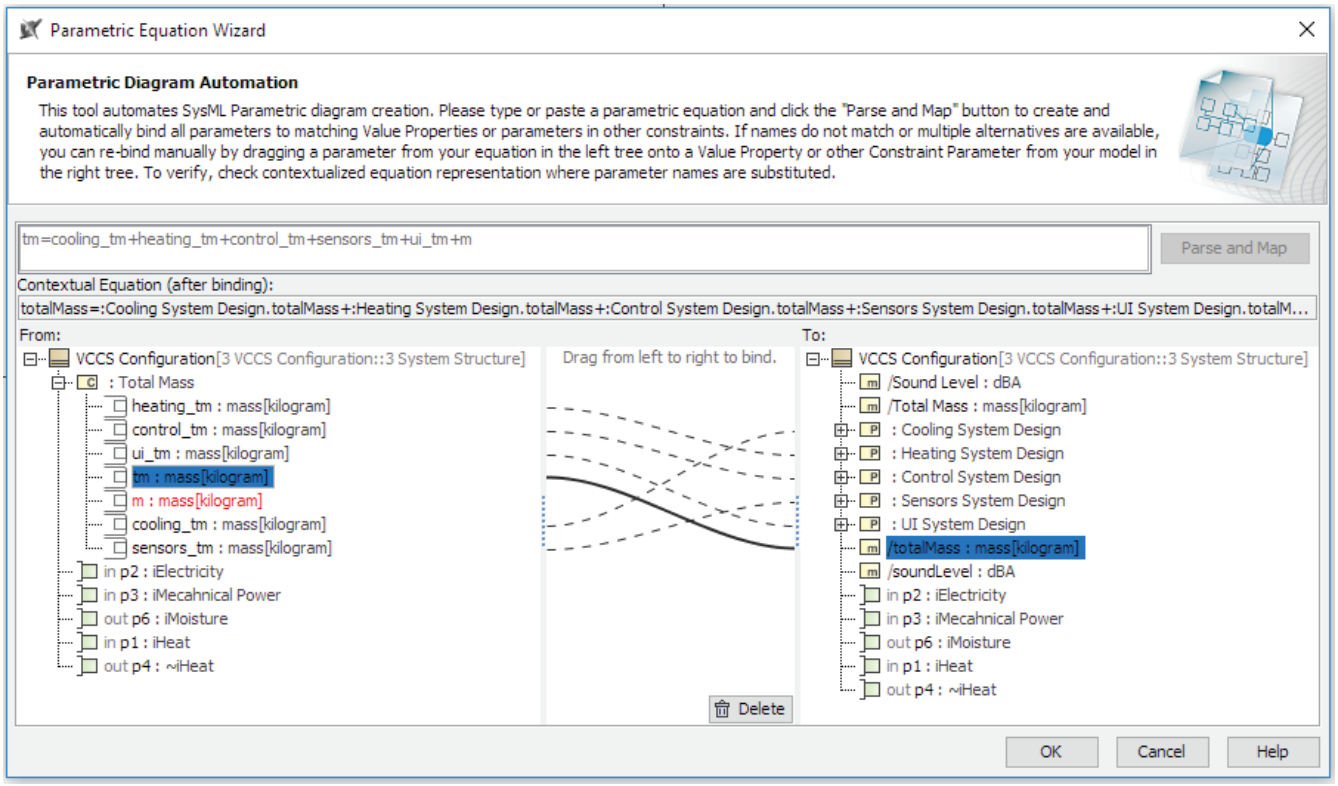
ⓘ As you can see, the value properties are displayed in the dot notation. If you want to switch to the nested notation, select all the shapes, except the one of the *totalMass* value property (the second shape on the top), then right-click them and select **Refactor** > **Convert to Nested Parts**.



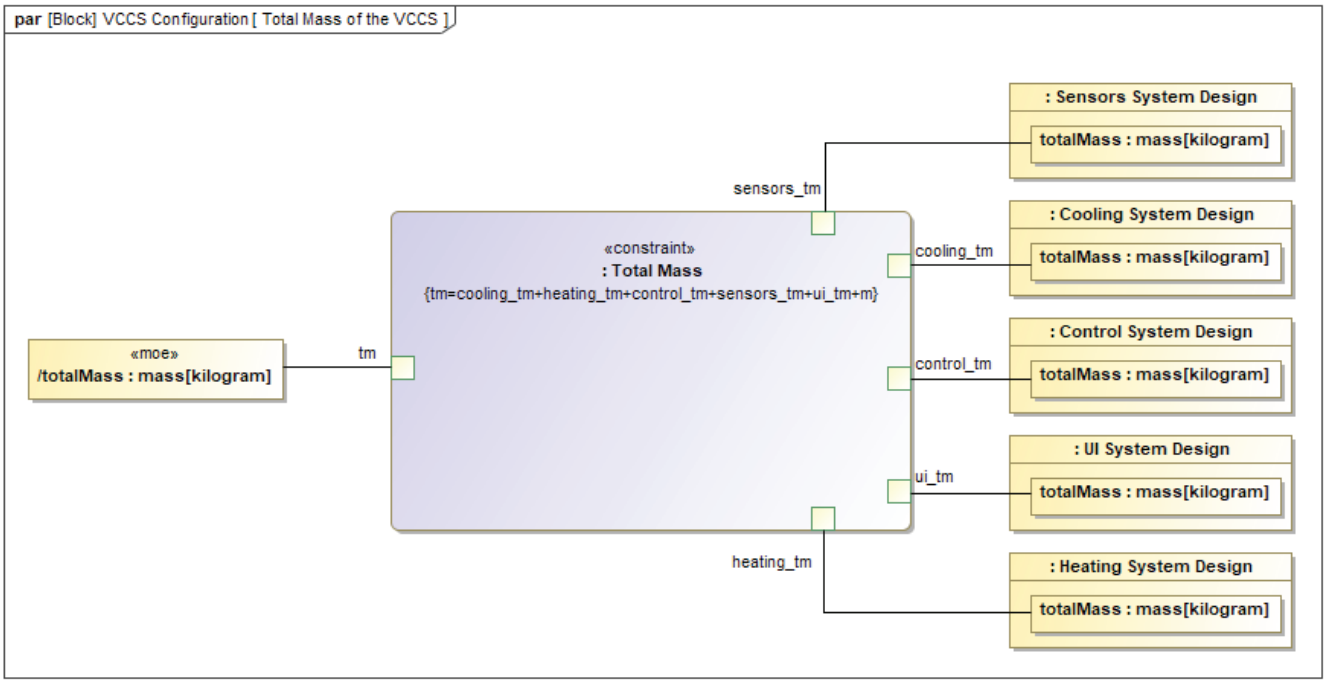
d. Type *Total Mass of the VCCS* to name the diagram and press Enter.

4. Find the *Total Mass* constraint block:
   a. Press Ctrl + Alt + F. The **Quick Find** dialog opens.
   b. Type *total*.
   c. When you see the *Total Mass* constraint block selected in the search results list, press Enter. The constraint block is selected in the Model Browser.
5. Drag the *Total Mass* constraint block to the *Total Mass of the VCCS diagram*. An unnamed constraint property is created for the *VCCS Configuration* block. The constraint property is typed by the *Total Mass* constraint block, which means that this constraint block is applied on the *VCCS Configuration* block. The **Parametric Equation Wizard** dialog opens concurrently.
6. Select constraint parameters (on the left side of the dialog) one by one and drag each to the corresponding value property (on the right side of the dialog):
   - *heating_tm* to the *totalMass* value property of the *Heating System Design* block
   - *control_tm* to the *totalMass* value property of the *Control System Design* block
   - *ui_tm* to the *totalMass* value property of the *UI System Design* block
   - *tm* to the *totalMass* value property of the *VCCS Configuration* block
   - *cooling_tm* to the *totalMass* value property of the *Cooling System* Design block
   - *sensors_tm* to the *totalMass* value property of the *Sensors System Design* block

   ⓘ • You may need to expand the structure on the right side of the dialog to see the value property you need. Remember that majority of them belong to the subsystem blocks, but not to the system block directly.
   • Once bound, the constraint parameter changes its color from red to grey.



7. After you bind all the items except the *m* constraint parameter (the corresponding value property is not created yet!), click **OK**. The dialog closes and all the bindings are displayed on the diagram pane.



There are a few more ways of establishing binding connectors. You can learn one of them to bind the *m* constraint parameter to a new value property that captures the mass of the VCCS itself.

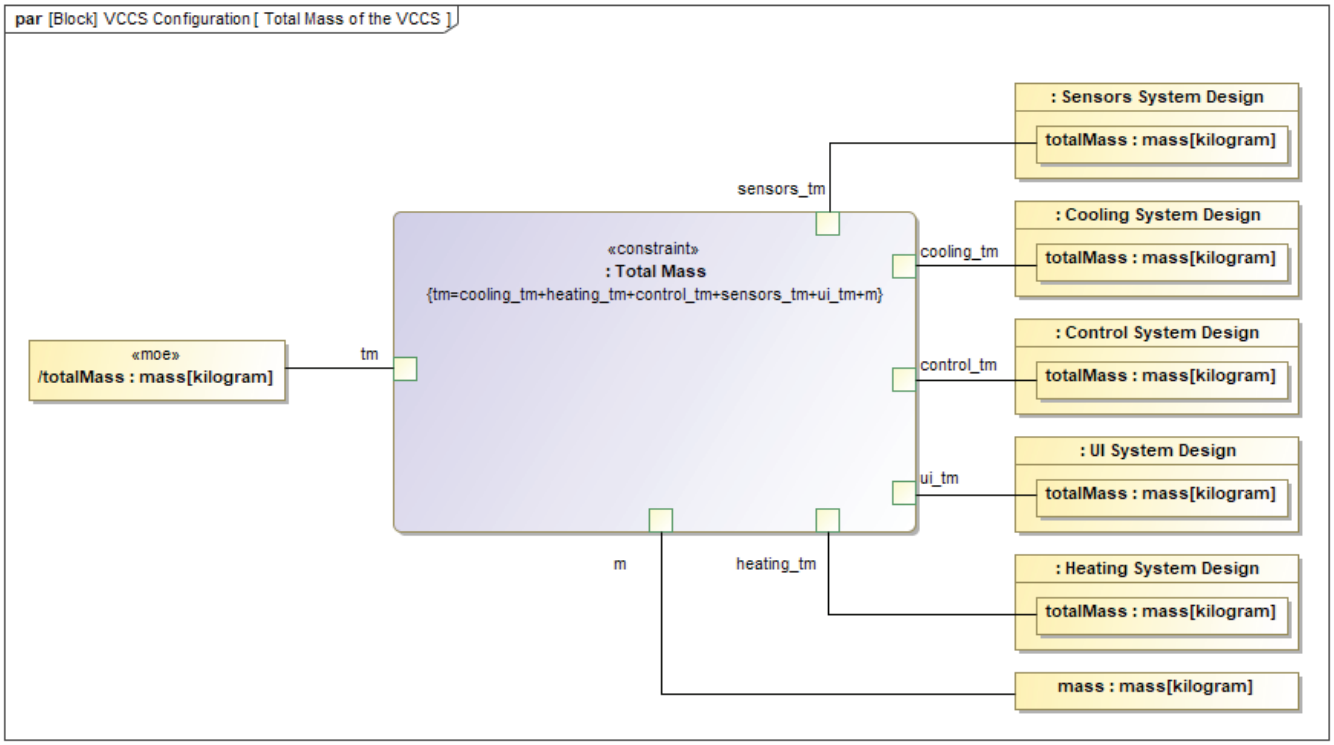
To bind the *m* constraint parameter to the new value property

1. Select the shape of the constraint property on the diagram pane and click the Display All Parameters button on its smart manipulator toolbar. The *m* constraint parameter is displayed on the shape of that constraint property.
2. Select the shape of the *m* constraint parameter and click the Binding Connector button on its smart manipulator toolbar.
3. Click somewhere you see good on the diagram pane. An unnamed value property is created.

   ⓘ Like the diagram, where the *mass* value property is displayed, it is owned by the *VCCS Configuration* block. To make sure, right-click the shape of the mass value property and choose **Select in Containment Tree**.

4. Directly on the shape, type *mass:mass[kilo* and press Ctrl + Spacebar.

5. Select the *mass[kilograms]* type from the list below and press Enter.



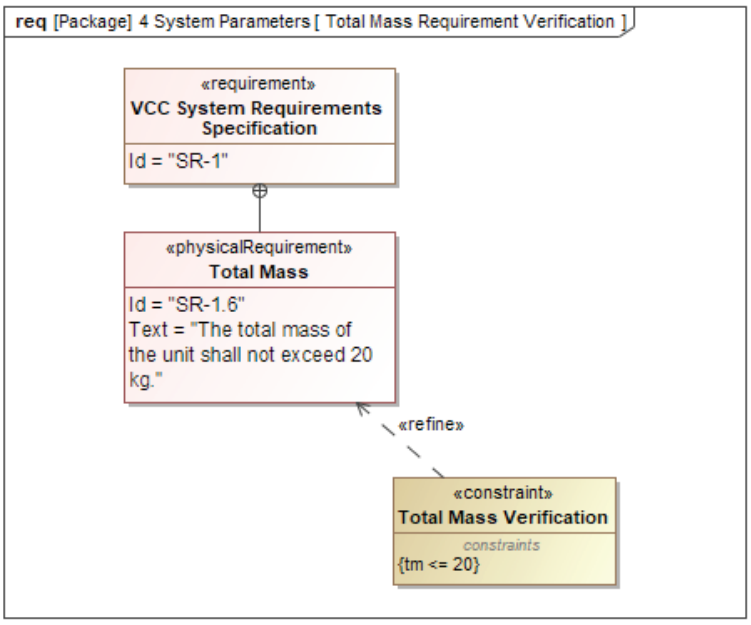
Now you can run the simulation, provide input values, and see the result of the calculation.

> ⓘ The simulation capabilities are not discussed here, since they concern the model usage, not the modeling. To learn more about the simulation, see the latest documentation of your modeling tool.

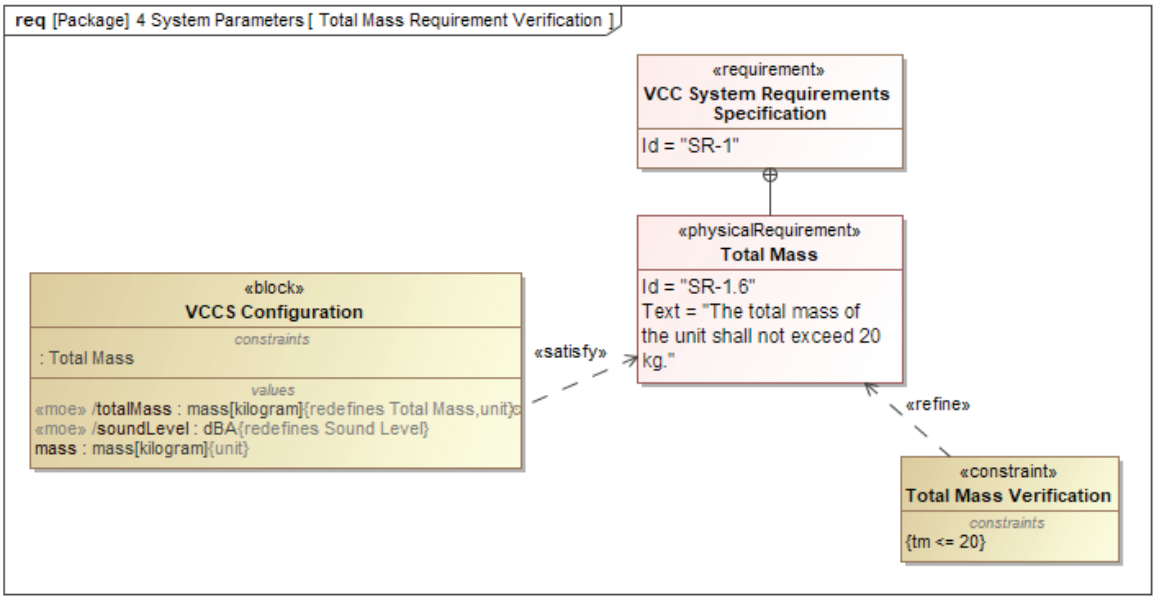
Step 6. Getting ready for the automated requirements verification

Having the system parameter calculated, you can verify the *Total Mass* system requirement and give the verdict on whether it is satisfied or not. The modeling tool enables you to perform this verification automatically. To get ready for the automated requirements verification, you need to:

1. Formalize the constraint captured as the natural language phrase in the text of the *Total Mass* system requirement. In terms of SysML, you should create another constraint block with another constraint expression, this time the *tm <= 20* inequality, and then draw a refine relationship between that constraint block and the *Total Mass* system requirement, as you can see in the following figure.



2. Indicate the system parameter, whose value determines whether the system requirement is satisfied or not. In terms of SysML, you should draw a satisfy relationship from the *totalMass* value property, which captures the system parameter, to the *Total Mass* system requirement.



Step 7. Adding more complexity to the calculations

The previous steps describe quite simple calculations, where all inputs are defined manually. However, situations like this are rare in real-world cases. Usually calculations are more complex, and outputs of one constraint expression become inputs for another constraint expression.

The formula specifying how to calculate the total mass of the VCCS can become more complex as well. It can be captured as a combination of several constraint expressions. You can specify adequate constraint expressions for each design subsystem of the VCCS and make their *totalMass* value properties calculable instead of being manually defined.

While the constraint block for calculating the total mass of the VCCS is defined in the VCCS configuration model, the constraint block for calculating the total mass of the Cooling System should be defined in the solution domain model of the Cooling System. Value properties that capture the mass values of Cooling System components should be defined in that model as well. By following the instructions given in the previous steps of this tutorial, you can update the solution domain model of the Cooling System with adequate elements.

ⓘ The total mass of the system can also be calculated by using the capabilities of the modeling tool. Instead of defining constraints and value properties manually, the **MassRollUpPattern** can be applied to the *VCCS Configuration* block. To learn more about roll-up patterns, see the latest documentation of the modeling tool.

S1. System Requirements: final

What is it?

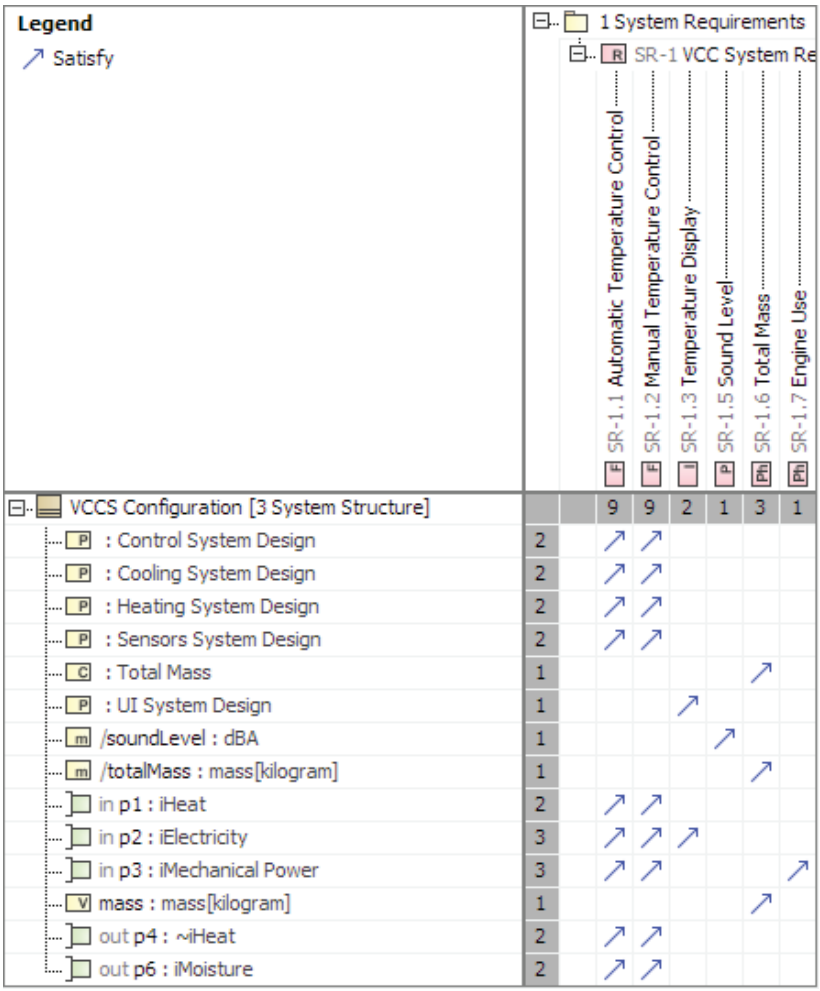
To have a complete solution domain model, you need to establish traceability relationships between the elements that capture the system requirements and the elements that capture the solution architecture of the system under design. As determined by the SysML, these are the satisfy relationships, meaning that one of the latter fulfills one of the former; for example, a satisfy relationship can be established from a block that captures some logical subsystem or component to certain system requirement. In an ideal case, all system requirements are satisfied by one or more elements from the solution domain model.

Who is responsible?

In a typical multidisciplinary systems engineering team, these traceability relationships should be established by the *Requirements Team*.

How to model?

Neither of the SysML diagrams is suitable for creating a mass of cross-cutting relationships, such as «satisfy». For this, a matrix or a map is more suitable. The modeling tool offers a wide variety of predefined matrices for specifying cross-cutting relationships. To capture «satisfy» relationships, a Satisfy Requirement Matrix can be used.



What's next?

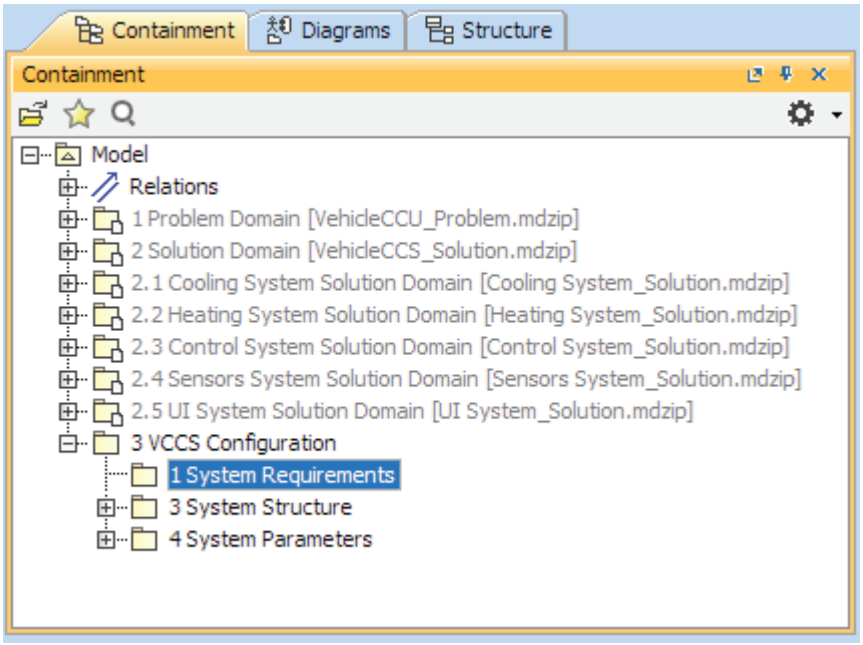
- Once you're done with the system requirements traceability, you're done with the solution architecture of the system under design. This means, you can switch to the Implementation domain.

Tutorial

Step 1. Organizing the model for S1 final

Let's assume that you want to establish traceability relationships between the system requirements and the elements that capture the solution architecture of the concrete system configuration. In such case, the Satisfy Requirement Matrix

should be created inside the *1 System Requirements* package, a subpackage of the *3 VCCS Configuration* package in the configuration model of the VCCS (the *VCCS_Solution.mdzip* file).



To organize the model for S1 final

1. Open the system configuration model of the VCCS (the *VCCS_Solution.mdzip* file) you created in step 1 of the S3 final phase tutorial, if not opened yet.
2. Right-click the *3 VCCS Configuration* package and select **Create Element**.
3. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.
4. Type *1 System Requirements* to specify the name of the new package and press Enter.

Step 2. Creating a matrix for capturing satisfy relationships

In this step, you should create a Satisfy Requirement Matrix and specify its criteria. System requirements can be displayed in the columns of the matrix, and the elements that capture the system configuration can be displayed in its rows. As opposed to a common dependency matrix, a Satisfy Requirement Matrix has the column elements type and dependency criteria predefined, which helps to speed up building the view.

To create a Satisfy Requirement Matrix

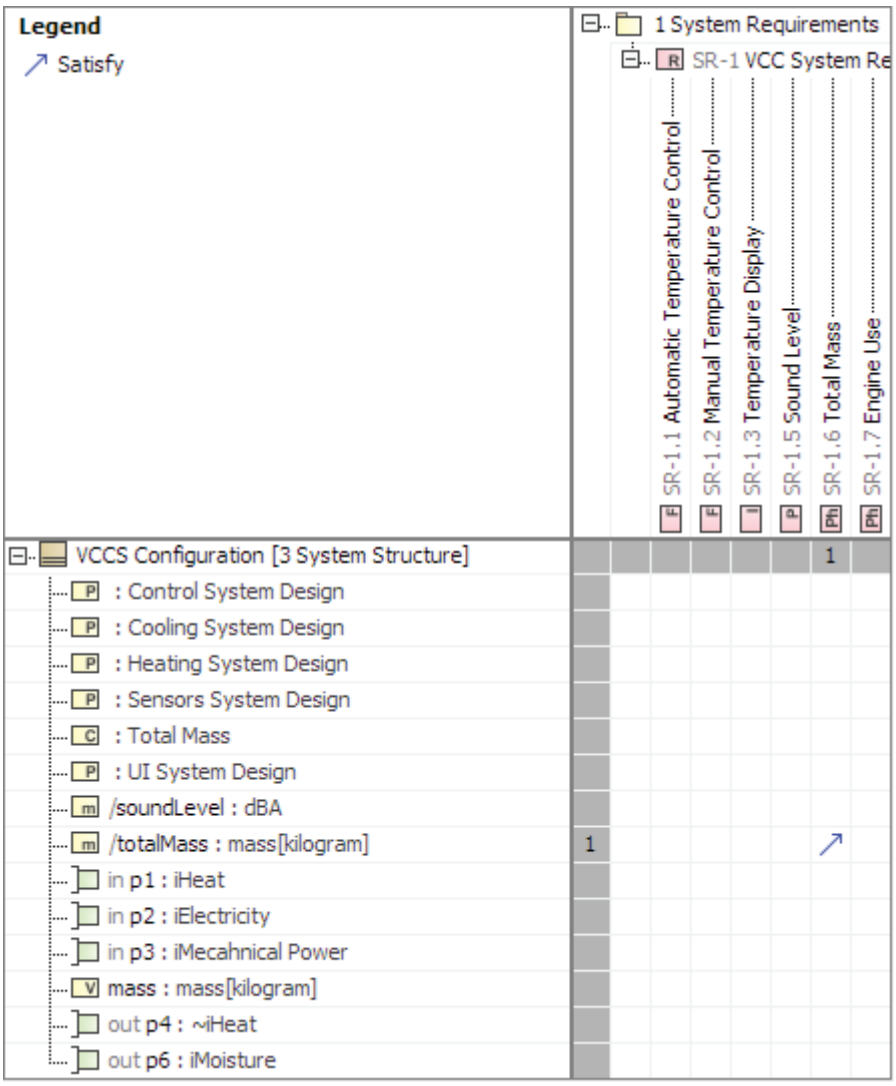
1. In the Model Browser, right-click the *1 System Requirements* package and select **Create Diagram**.
2. In the search box, type srm, the acronym of the predefined Satisfy Requirement Matrix, and press Enter.

   ⓘ If you don't see any result, click the **Expert** button below the search results list. The list of available diagrams expands in the Expert mode.
3. Type *System Configuration to System Requirements* to specify the name of the new matrix and press Enter again.
4. In the Model Browser, expand the *2 Solution Domain* package (if not yet expanded), select the *1 System Requirements* package, and drag it onto the **Column Scope** box in the **Criteria** area above the matrix contents. The *1 System Requirements* package, an inner package of the *2 Solution Domain* package, becomes the scope of the matrix columns.
5. In the Model Browser, select any part property, proxy port, constraint property, and value property with the «moe» stereotype and drag the selection onto the **Row Element Type** box in the **Criteria** area above the matrix contents.

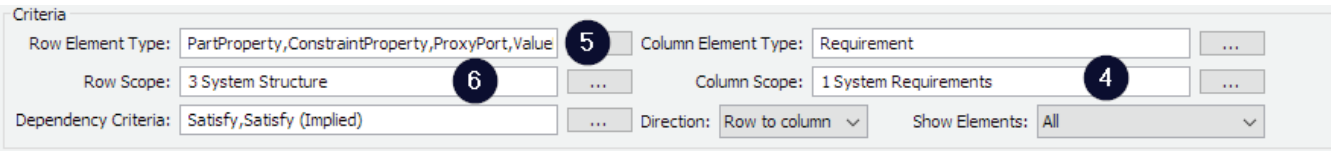
Part properties, proxy ports, constraint properties, and value properties (with the «moe» stereotype) become types of the matrix rows.

> ⓘ To select the set of non-adjacent items in the tree, click the first one, press Ctrl, and while holding it down, select other items one by one. This also applies to step 6.

6. In the Model Browser, directly under the *4 VCCS Configuration* package, select the *3 System Structure* package, a sibling package of the *1 System Requirements* package with the Satisfy Requirement Matrix, and then drag the selection onto the **Row Scope** box in the **Criteria** area above the matrix contents. The *3 System Structure* package becomes the scope of the matrix rows, and the contents of the matrix is updated. All the cells are empty at the moment, except the one at the intersection of the *totalMass* value property and the *SR-1.6 Total Mass* requirement (remember that this relationship was created in step 6 of the S4 tutorial).



The following figure displays the **Criteria** area of the matrix, with highlights on step 4, 5, and 6 related criteria.

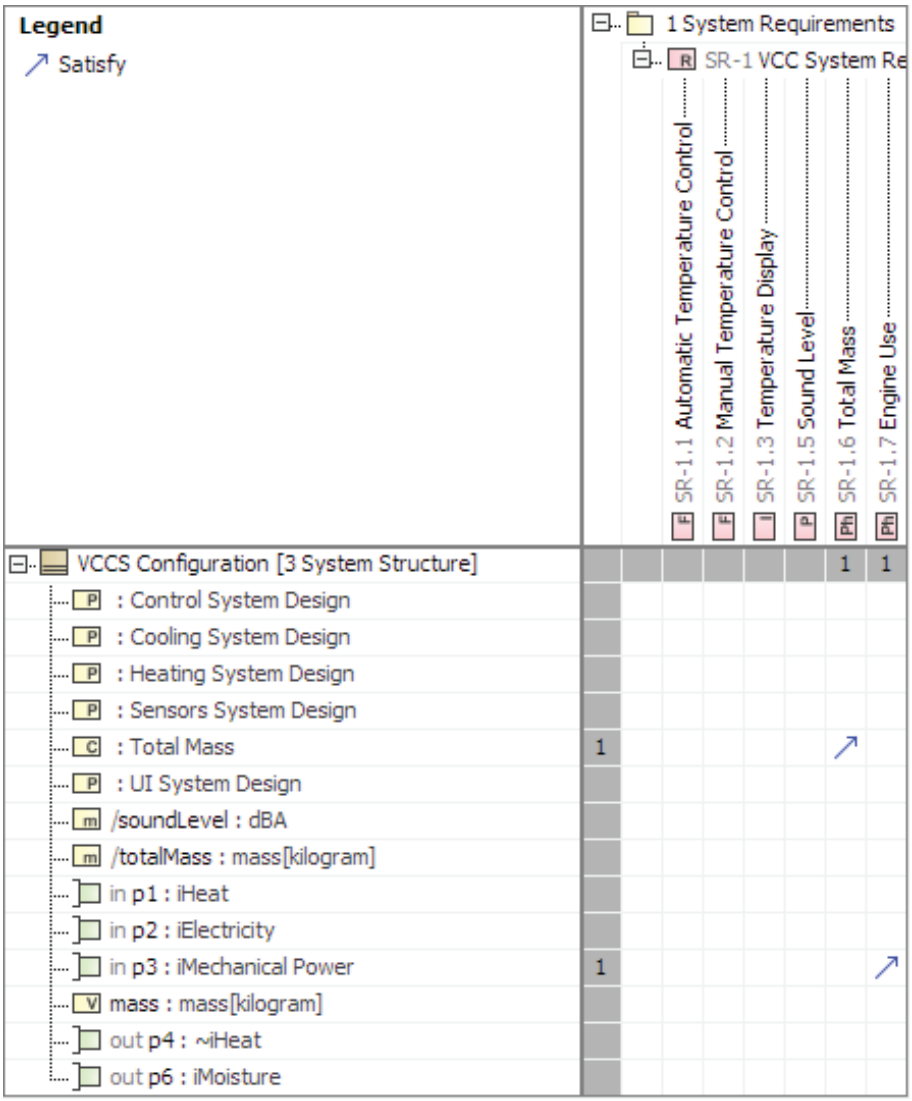


Step 3. Capturing satisfy relationships

Now you're ready to establish more satisfy relationships. Let's specify that the proxy port typed by the *iMechanicial Power* interface block satisfies the *SR-1.7 Engine Use* requirement.

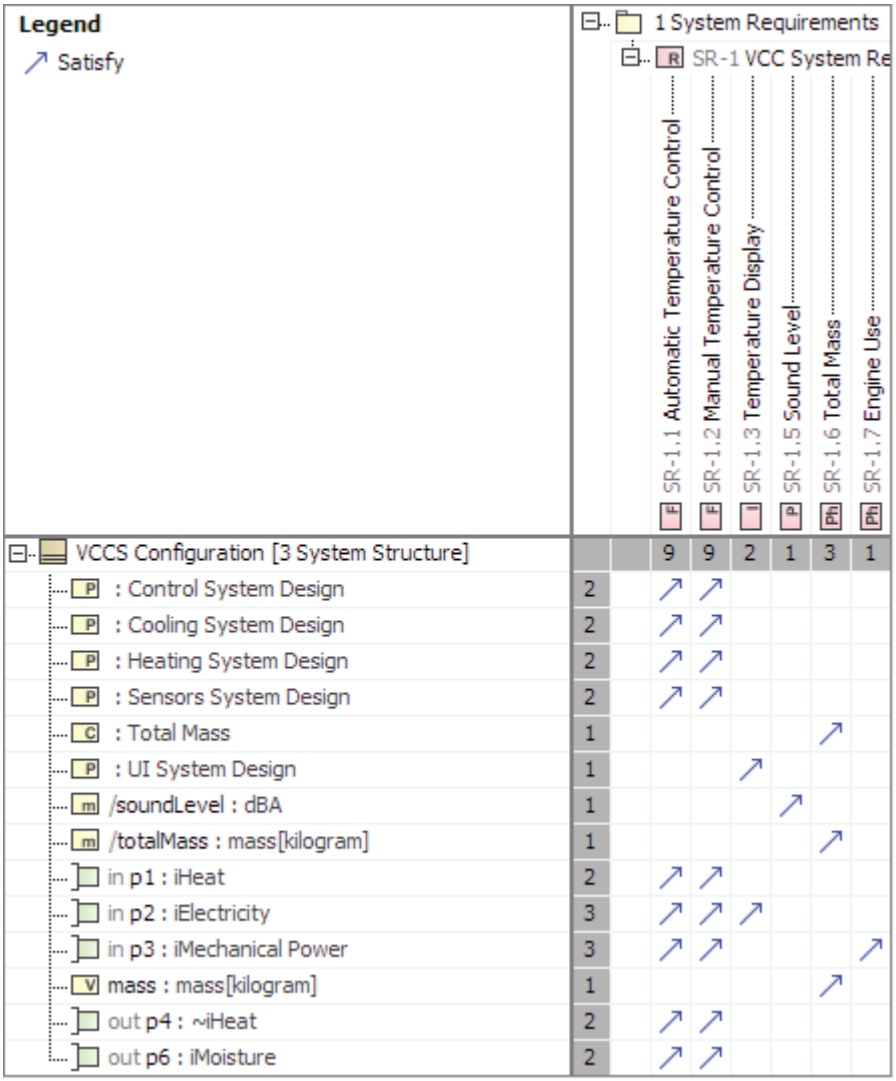
To specify the satisfy relationship in the matrix

- Double-click the cell at the intersection of the row that displays the proxy port typed by the *iMechanicial Power* interface block and the column that displays the *SR-1.7 Engine Use* requirement. The satisfy relationship is created between the appropriate items and displayed in the cell.



After all satisfy relationships are established, your *System Configuration to System Requirements* matrix should resemble the one in the following figure. In the ideal case, every system requirement must be satisfied by one or more elements

from the system configuration model. The final version of the matrix tells the systems engineer how well the system configuration satisfies the system requirements.



Implementation domain

Introduction

After the solution architecture of the whole system is defined and the optimal system configuration is selected (by means of the trade-off an alysis), the system is ready for implementation. At this point in time you should start thinking of the system as real, not abstract any longer, like you did in the problem and solution domains.

As you can see in the following table, the implementation domain is only partially covered by the MagicGrid framework. The approach defines physical requirements specification of the system under implementation, but its detailed design is not a part of MBSE, nor a part of the MagicGrid approach. The detailed design of the system is a part of the model-based design (MBD) and can be developed by using tools like electrical and mechanical CAD software or automated code generation tools, to name a few.

	PILLAR						
DOMAIN			**Requirements**	**Behavior**	**Structure**	**Parameters**	
	Problem	**Black Box**	B1-W1 Stakeholder Needs	B2 Use Cases	B3 System Context	B4 Measurements of Effectiveness	**Specialty Engineering**
		White Box		W2 Functional Analysis	W3 Logical Subsystems Communication	W4 MoEs for Subsystems	
	Solution		S1 System Requirements	S2 System Behavior	S3 System Structure	S4 System Parameters	**Integrated Testing**
			SS1 Subsystem Requirements	SS2 Subsystem Behavior	SS3 Subsystem Structure	SS4 Subsystem Parameters	
			...	...	...	...	**Analysis**
			C1 Component Requirements	C2 Component Behavior	C3 Component Structure	C4 Component Parameters	
	Implementation		I1 Physical Requirements	Software, Electrical, Mechanical			

I1 Physical Requirements

What is it?

To be able to implement a system, you first need to have the detailed physical requirements of that system in order to follow them while producing a detailed design of it. The detailed physical requirements are based on the solution architecture of the system under design, including the solution architectures of all its subsystems and even more specific parts.

The physical requirements specification is given to the engineers of various disciplines, who are responsible for designing different aspects of the system under implementation.

Who is responsible?

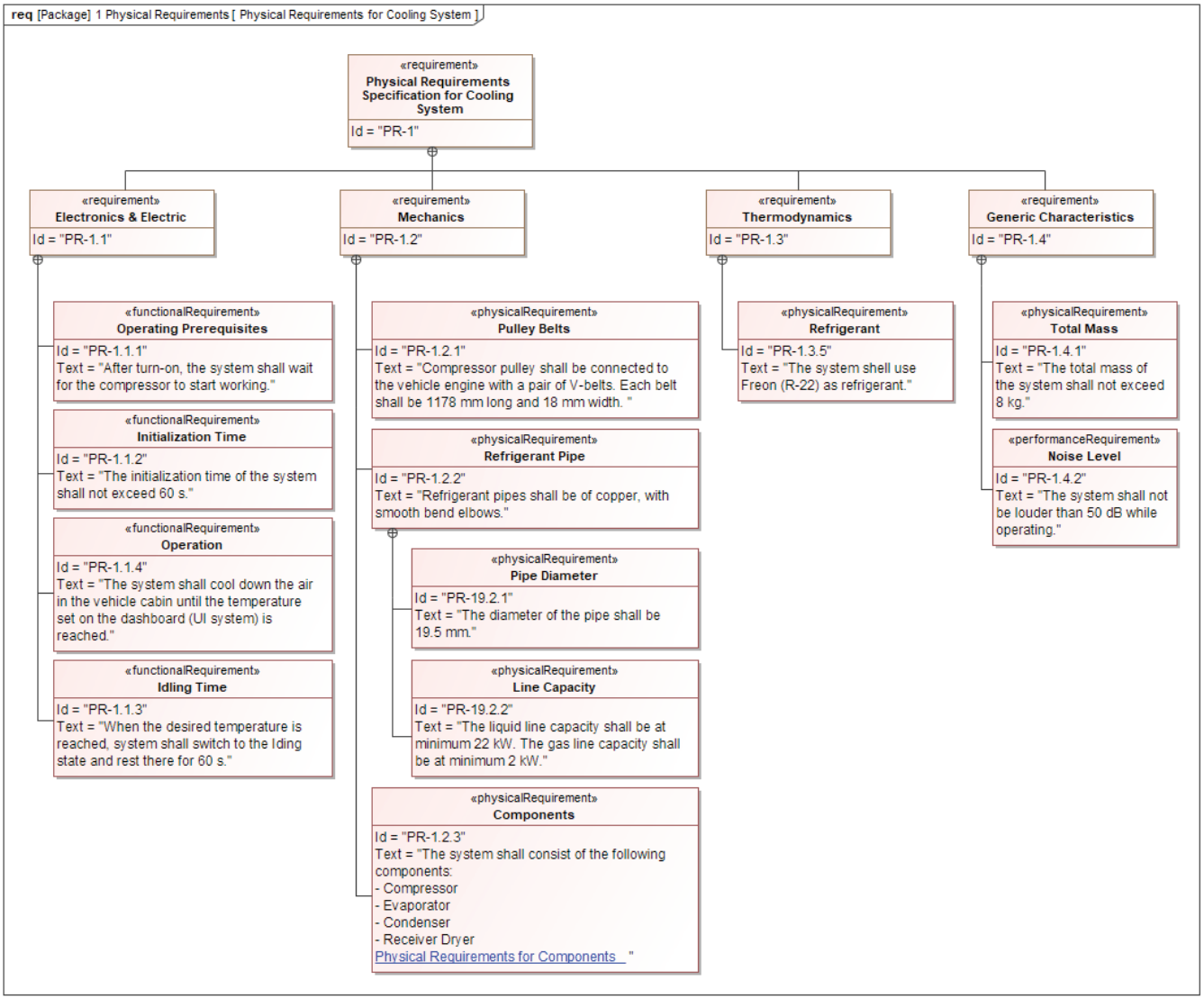
In a typical multidisciplinary systems engineering team, the physical requirements of the system under implementation are specified by the *Requirements Team*.

How to model?

In the modeling tool, an item of the physical requirements specification can be stored as a SysML requirement, which has a unique identification, name, and textual specification. Physical requirements must be formal and written down by following certain guidelines, for example; write in short and clear sentences only; avoid conditional keywords, such as

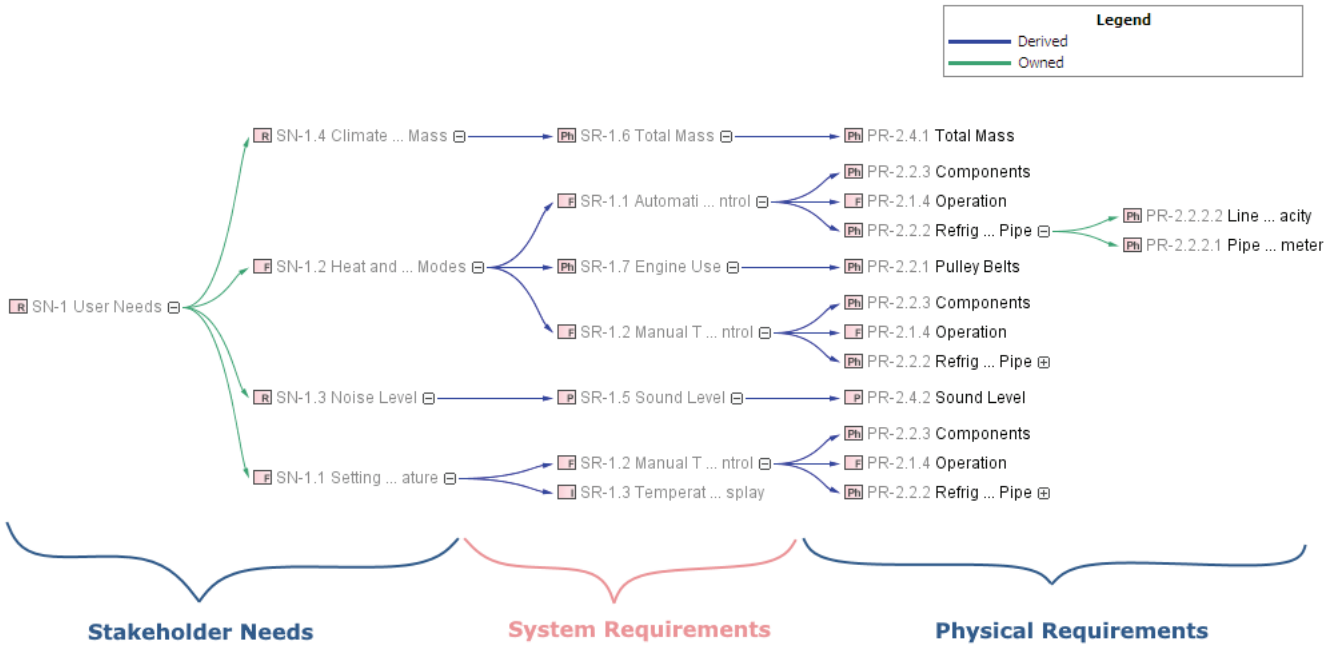
if, except, as well; use the *shall* keyword; and include tables. For more information on how to write good textual requirements, refer to *INCOSE Systems Engineering Handbook*.

Managing physical requirements is similar to managing stakeholder needs and system requirements. Just like them, physical requirements can be categorized and organized into groups, subgroups, and so forth. SysML supports a wide variety of requirement categories, such as physical, functional, and performance, to name a few. For grouping requirements, use the containment relationships among them. The SysML requirements that capture grouping items might not have any text itself. As you can see in the following figure, which shows an example of physical requirements for the Cooling System, there are four groups of requirements: *Electronics & Electric, Mechanics*, *Thermodynamics*, and *Generic Characteristics*. You can also see that all items are categorized into functional, physical, and performance requirements.

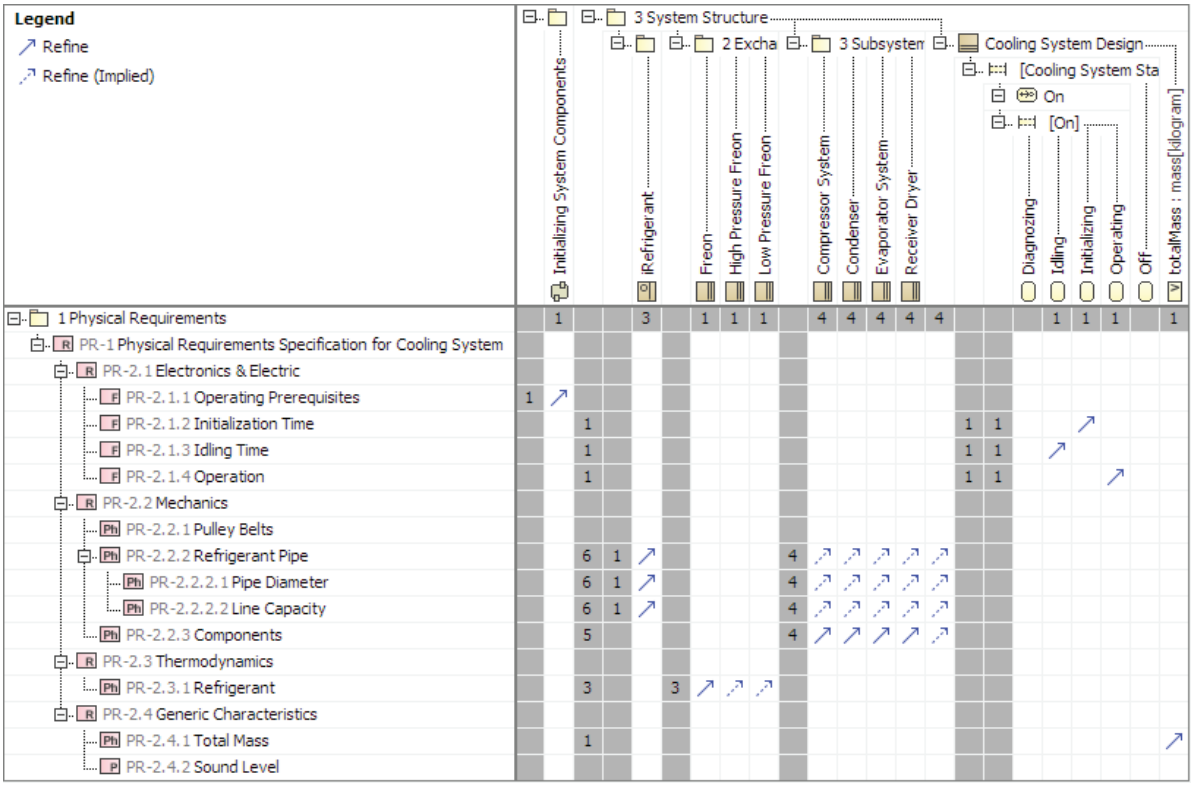


Physical requirements should be derived from the system, subsystem, or component requirements defined in the solution domain. Each physical requirement must cover at least one item of the system, subsystem, or component requirements. For this, you can use the «deriveReqt» relationship pointing from the physical requirement to that item of the system, subsystem, or component requirements. Having derivation relationships among requirements in different domains, you can easily trace from the very detailed requirements to the very abstract requirements and see what items determine the creation of other items. For performing such impact analysis (either downstream or upstream), we recommend utilizing

the infrastructure of the Requirement Derivation Map, one of the predefined relationship maps in the modeling tool (you can see an example in the following figure).



Moreover, each physical requirement should refine one or more elements from the solution domain model. For this, you can use the «refine» relationship pointing from the physical requirement to some structural or behavioral element within the solution domain model. For establishing and reviewing these relationships, we recommend utilizing the infrastructure of the Refine Requirement Matrix, one of the predefined dependency matrices in the modeling tool (the example is displayed in the following figure). As you can see, the physical requirements specification is not yet completed, since there are some elements in the solution domain that are not yet refined by any physical requirement; for example, the *Diagnosing* and *Off* states. Also, it shouldn't happen in a real-world case that a physical requirement refines no element from the solution domain; for example, the *PR-2.2.1 Pulley Belts* and *PR-2.4 Sound Level* requirements. In such case, either the solution domain model must be revised and updated, or the physical requirements specification must be reviewed and improved to conform to the solution domain model.



Tutorial

Steps of the I4 tutorial are adequate to the steps of the S1 initial phase tutorial:

- Step 1. Creating and organizing a model for S1 initial
- Step 2. Creating a diagram for system requirements
- Step 3. Specifying system requirements
- Step 4. Establishing traceability to stakeholder needs
- Step 5. Establishing traceability to the rest of the problem domain model

What's next?

- When you have the physical requirements of the system under implementation, you are ready to develop the detailed design of the system. It's time to switch to the model-based design in order to focus on different aspects of the system, such as mechanical, electrical, and software components, to name a few. These activities are outside the scope of the MagicGrid approach.

Future works

Thank you for taking time to read the first edition of the ***MagicGrid BoK.*** Your feedback on the book is more than welcome. All comments will be taken into account before releasing the second edition of the book. Moreover, everyone of you can sign-up as a reviewer for the second edition of the ***MagicGrid BoK***. Please express your interest by sending an email to magicgrid@nomagic.com.

Besides the comments we will get, we have plans to extend the scope of the ***MagicGrid BoK.*** The first edition of the book is the starting point for us to move the MBSE practicing effort to a much broader scope. The main focus of this book is to introduce readers to a new, practically proven approach of developing the system model. Model development, however, is only one of many aspects of modelbased systems engineering. There are other aspects that we want to address in future editions of the ***MagicGrid BoK***, such as model governance and model usage, to name a few.

Glossary

Action
A SysML element that can capture a basic unit of the system functionality within the activity. It represents some form of processing or transformation that will occur when the activity is executed during the system operation. An action is represented as a round-cornered rectangle, with the name displayed as a string inside that rectangle.

Activity
A type of behavioral SysML element that can contain a set of model elements, such as actions and flows, and the activity diagram that represents these elements.

Activity diagram
A type of behavioral SysML diagram that can be used to describe the flow of control or the flow of inputs and outputs among actions. Activity diagrams can be used to specify the black- and whitebox scenarios of the expected system functionality, and in combination with state machine diagrams to design the system behavior.

Actor
A SysML element that can capture a role played by a person, an organization, or another system when it interacts with the system of interest in the particular context. However, the MagicGrid approach recommends using a block instead of the actor to capture these concepts, for the reasons described in B3.initial.

Association
A type of SysML relationship that enables you to specify what actors interact with the system of interest in the particular system context to invoke or participate in a use case. The notation for the association is a solid line.

Binding connector
A special type of SysML connector that is used to represent an equality relationship between a constraint parameter and a value property in a parametric diagram, in order to convey that the constraint parameter receives the values from that value property. The notation for the binding connector is a solid line.

Black box / white box
The black box represents an external view of the system. During the system analysis from the blackbox perspective, the system is viewed in terms of its inputs and outputs, without any knowledge of its internal structure and behavior.

The white box represents an internal view of the system. During the system analysis from the whitebox perspective, the functional decomposition of the system is performed in order to identify functional blocks, also known as logical subsystems or components of the system.

Block
A SysML element of definition which can capture a basic unit of the system structure. A block is represented as a rectangle with a stereotype block preceding the name of the «block» in the name compartment.

Block definition diagram (bdd)
A type of structural SysML diagram that can be used to define features of blocks and relationships between blocks, such as associations, generalizations, and dependencies. It captures the definition of blocks in terms of properties and operations, and relationships such as a system hierarchy or a system classification tree. Block definition diagrams can be used to specify the structure of the system or subsystem, and to define the interfaces and constraints of that system or subsystem.

Call behavior action
A special type of SysML action that can invoke another behavior: an activity, a state machine, or an interaction. Call behavior actions enable you to decompose a higher-level behavior into a set of lower-level behaviors.

Composite association / Directed composition

A special type of SysML association that enables you to specify that one block is a part of another block. The notation for the composite association is a solid line between two blocks with a black diamond on the composite end and an open arrowhead on the part end. Drawing a composite association between blocks creates a part property for the block that appears on the black diamond end of the relationship; the part property can have a name and is typed by the block which appears on the arrowhead end of the composite association.

Component

An elementary object of the system structure. Components are often grouped into subsystems of the system; they are identified by decomposing these subsystems. A component can be captured as a block in the model.

Connector

A type of SysML relationship that is used to represent communication between two part properties in an ibd. The connector may be established between part properties through various kinds of ports (as long as these ports are compatible); for example, two part properties connected through proxy ports convey that they exchange some matter, energy, or data that can flow between them over the connector, through compatible proxy ports. The notation for the binding connector is a solid line.

Constraint block

A SysML element of definition that can capture a constraint expression (either an equation or an inequality). The constraint expression enables you to constrain the value properties of blocks. The variables of the constraint expression are called constraint parameters. These constraint parameters receive their values from the value properties that are being constrained. The notation of the constraint block is the same as for the block, but with the «constraint» stereotype before the name of the element.

Constraint parameter

A variable that appears in a constraint expression of the constraint block. It is captured as a property of the constraint block in the model, and is represented as a string in the parameters compartment of that constraint block.

Constraint property

A usage of a constraint block in the context of some block or, in other words, a property of a block typed by a constraint block defined somewhere in your model. It can be represented as a string in the constraints compartment of that block in a bdd, or as a round-cornered rectangle in a parametrics diagram. The name of the constraint property in both cases is displayed as a string followed by a colon and the name of the constraint block which types that property; for example, mass : *totalMass*.

Control flow

A type of SysML relationship between a pair of actions. Control flows enable you to express the order in which actions are performed in the associated activity. Control flow is represented as an dashed line with an open arrowhead.

Decision/Merge

The decision node marks the start of alternative flow in an activity. It is represented as a hollow diamond, and must have a single incoming flow and two or more outgoing flows.

The merge node marks the end of the alternative flow in an activity. It is represented as a hollow diamond, and must have two or more incoming flows and a single outgoing flow.

Exchange item

A concept that defines a type of matter, energy, or data that can flow between two structural objects within a system; for example, between two part properties that capture the subsystems of the system, in an ibd. An exchange item can be captured

as a block or a signal and then assigned as an item flow to the connector. In such case, the connector is decorated with a filled-in triangle.

Flow property

A property of an interface block. It can have a direction and a type. The direction of the flow property specifies whether the property represents an input or an output of the structural object within the model.

Fork/Join

The fork node marks the start of concurrent flows in an activity. It is represented as a line segment (either horizontal or vertical), and must have a single incoming flow and two or more outgoing flows.

The join node marks the end of concurrent flows in an activity. It is represented as a line segment (either horizontal or vertical), and must have two or more incoming flows and a single outgoing flow.

Function

A task performed by a system, or a certain part of it, to transform inputs to outputs. A single function can be captured as an activity in the model.

Functional block

A concept that is used to define a structural object which is responsible for performing one or more functions of the system. Functional blocks are identified by performing the functional decomposition of the system. Functional blocks serve as inputs for defining the structure of the system of interest. They can also be called logical subsystems or logical components in different levels of the functional decomposition.

Functional decomposition / Functional breakdown

A set of steps to break down the overall function of a system into its smaller parts, in order to identify functional blocks.

Graphical User Interface (GUI)

A type of user interface that allows the use of icons or other visual indicators to interact with electronic devices, rather than using only text via the command line.

High-Level Solution Architecture (*HLSA*) model

The core of a solution domain model. Based on the results of the problem domain analysis, it defines the subsystems of the system under design (as a single-level hierarchy). The purpose of the *HLSA* model is to identify work packages, one for each subsystem, and allocate them to separate engineering teams. For more information, refer to Solution domain.

Interface

A hardware or software component that connects two or more structural objects of the system or the system and the objects from its environment, for the purpose of passing matter, energy, or data from one to the other.

Interface block

A SysML element of definition that can contain a set of flow properties which define the inputs and outputs of some structural object captured in the model as a block; for example, a system, a subsystem, or a component. The notation of the interface block is the same as for the block, but with the «interfaceBlock» stereotype before the name of the element.

Internal block diagram (ibd)

A type of structural SysML diagram that can be used to capture the internal structure of a block in terms of properties and connectors between these properties. Block definition diagrams can be used to specify the system context and the interactions within the system.

Measurement of Effectiveness (MoE)

A characteristic of the SoI in numerical format, MoE is a traditional term widely used in systems engineering, and describes how well a system carries out a task within a specific context.

Model Browser
An element of the modeling tool GUI which represents the contents of the model repository. It is by default located at the left of the main window of the modeling tool.

Model-based design (MBD)
A mathematical and visual method of addressing problems associated with designing complex control, signal processing, and communication systems.

Model-Based Systems Engineering (MBSE)
The formalized application of modeling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases. MBSE focuses on creating and exploiting models as the primary means of information exchange between engineers, rather than documentbased information exchange.

Modeling tool
Either Cameo Systems Modeler or MagicDraw with the SysML Plugin installed on top.

Object flow
A type of SysML relationship between a pair of actions. Object flows enable you to express what matter, energy, or data flows through an activity from one action to another when the activity executes. Object flow is represented as a solid line with an open arrowhead.

Package
A kind of SysML element that can contain other elements. *Package*s can be used to organize the elements into logically cohesive groups.

Parametric diagram
A special type of ibd which displays the internal structure of a block, with focus on bindings between the value properties of that block and the constraint parameters of the applied constraint block. Parametric diagrams can be used to calculate the values of system parameters (by utilizing the simulation capabilities of the modeling tool).

Part property
A usage of a block in the context of another block, or in other words, a property of a block typed by another block that is defined somewhere in your model. It can be represented as a string in the parts compartment of that block in a bdd, or as a rectangle in an ibd. Using part properties enables you to specify the internal structure of the system captured in the model as a block. A simplified ibd (without proxy ports) can be used to specify the interactions within a system context; in such case the system context is captured as a block, with a part property typed by the block that captures the system of interest.

Presentation artifact
A visual representation of some fragment or aspect of the system model. It can be a diagram, matrix, map, or table. Presentation artifacts work as data inputs to the model or data editors.

Proxy port
A type of property of the block, typed by an interface block defined somewhere in your model. It represents the point at the boundary of the structural object represented by that block, through which the external entities can interact with that structural object. In other words, it is a usage of an interface block in the context of another block. A proxy port is always represented as a small rectangle on the shape of the block or part property, usually decorated with an arrow to indicate whether the proxy port is for accepting inputs to the system, subsystem, or component, or providing outputs.

Requirement
A SysML element that can capture a text-based statement that translates or expresses a need. It can be used to capture stakeholder needs, or to specify the system requirements and detailed physical requirements in the model.

Sequence diagram

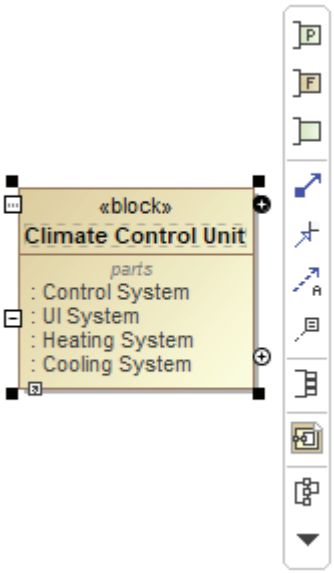
A type of behavioral SysML diagram that can be used to describe how the parts of a block interact with one another via operation calls and asynchronous signals.

Signal

A SysML element that you can use to capture an exchange item.

Smart manipulator toolbar

A piece of the modeling tool GUI. It appears when a symbol of the element is selected on the diagram pane.



Stakeholder

An individual or organization having a right, share, claim, or interest in a system or in its possession of characteristics that meet their needs and expectations.

Stakeholder needs

Information gathered from various stakeholders of the system of interest. This information includes primary user needs, system-related government regulations, policies, procedures, and internal guidelines, to name a few. A single stakeholder need can be captured as a requirement in the model.

State

An element that captures a state of the system, in which it can exist during the operation. A state is represented as a round-cornered rectangle with the name displayed as a string inside that rectangle.

State machine diagram

A type of behavioral SysML diagram that can be used to specify how a structure within a system changes state in response to event occurrences over time. It can be used to specify the behavior of the system or its parts.

Subsystem

A secondary or subordinate system within a larger system. A subsystem can be captured as a block in the model.

System context

A concept that determines an external view of the system. It must introduce elements that do not belong to the system, but do interact with it. Besides the system itself (considered to be a black box), the collection of elements in the particular system context can include external systems and users (humans, organizations) that interact with the SoI in that context. A single system context can be captured as a block in the model.

System

A combination of interacting elements that are organized into a complex whole for one or more common purposes.

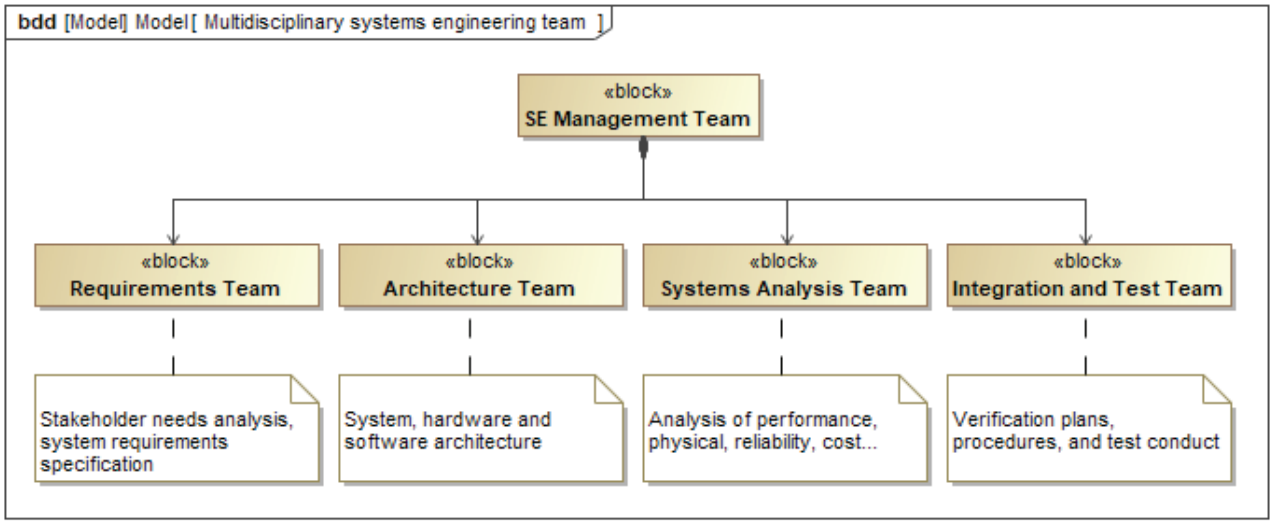
System of interest (SoI)

The system whose life cycle is under consideration. The system that is being analyzed first from the black- and then from the white-box perspective in order to produce the system requirements specification.

Systems Engineering (SE)

Interdisciplinary tasks which are required throughout a system's life cycle to transform stakeholder needs, requirements, and constraints into a system solution.

Systems Engineering Team



System requirements

Requirements to be met by the design of a system, subsystem, or component. System requirements are derived from stakeholder needs, which are more abstract requirements.

System under design

The system that is being designed in order to produce the physical requirements specification for the detailed system design.

System under implementation

The system whose detailed design is being developed.

System parameter

A numerical characteristic of the system, subsystem, or component. It can be specified as a value property of the block that captures that system, subsystem, or component. System parameters are derived from MoEs and can be verified against system requirements.

Systems Modeling Language (SysML)

A general-purpose modeling language for systems engineering applications. It supports the specification, analysis, design, verification, and validation of a broad range of systems and systemsof-systems.

Trade-off

A situational decision that involves diminishing or losing one quality, quantity, or property of a set or design in return for gains in other aspects. In simple terms, a trade-off is where one thing increases and another must decrease.

Transition

A type of SysML relationship that represents a change from one state to another. It can be triggered by some event concurrency.

Use case

A type of SysML element that can be used to capture what capabilities people expect from the system and what they want to achieve by using it within the particular system context. The notation for the use case is an ellipse with the name (usually a verb phrase) inside that ellipse.

Use case diagram

A type of behavioral SysML diagram that can be used to define a set of use cases performed within a particular system context; it represents the black-box view of the system of interest. It also displays and can be used for creating associations between the use cases and the participants of the system context, to specify who/what is responsible for invoking or participating in what use case.

Value property

A property of a block which can represent a quantity, a Boolean, or a string. It is displayed as a string in the values compartment of the block, and can be used to capture measurements of effectiveness of the system of interest (the «moe» stereotype must be applied in addition).

Bibliography

1. Chami M., Oggier, P., Naas, O., Heinz, M. “Real World Application of MBSE at Bombardier Transportation”, SWISSED, Zurich, 2015. Available at https://goo.gl/zf5uvp.
2. Delp, C., Lam, D., Fosse, E., Cin-Young Lee, “Model based document and report generation for systems engineering”, Aerospace Conference, IEEE, 2013. Available at http://www.omgsysml.org/ View_Paper-IEEE_2013.pdf.
3. Estefan, J. INCOSE Survey of MBSE Methodologies, INCOSE TD 2007-003-02, Seattle, WA, USA, 2008.
4. Friedenthal, S., et al. A Practical Guide to the SysML, 4th Edition: The Systems Modeling Language. Boston: MK/OMG Press, 2011.
5. Friedland, B., Herrold, J., Ferguson, G., Malone, R. “Conducting a Model Based Systems
6. Engineering Tool Trade Study Using a Systems Engineering Approach”, 27th Annual INCOSE International Symposium, pp. 1087-1099, Adelaide, 2017.
7. Hallqvist, J., Larsson, J. “Introducing MBSE by using Systems Engineering Principles”, 26th Annual INCOSE International Symposium, pp. 512–525, Edinburgh, 2016.
8. Hoffmann, H.-P. “Systems Engineering Best Practices with the Rational Solution for Systems and Software Engineering”, Release 3.2.1, February 2011. Available at https://www.slideshare.net/ billduncan/ ibm-rational-harmony-deskbook-rel-312.
9. ISO/IEC/IEEE 15288:2015 “Systems and Software Engineering–System Life Cycle Processes”, Geneva, 2015.
10. Mazeika, D., Morkevicius, A., Aleksandraviciene, A. “MBSE driven approach for defining problem domain”, 11th Conference of System of Systems Engineering (SoSE), pp.1-6, Kongsberg, 2016.
11. Morkevicius, A., Aleksandraviciene, A., Mazeika, D., Bisikirskiene, L., Strolia, Z. “MBSE Grid: A Simplified SysML- Based Approach for Modeling Complex Systems”, 27th Annual INCOSE International Symposium, pp. 136-150, Adelaide, 2017.
12. Morkevicius, A., Bisikirskiene, L., Jankevicius, N. “We Choose MBSE: What’s Next?”, Proceedings of the Sixth International Conference on Complex Systems Design & Management, CSD&M 2015, pp. 313, Paris, 2015.
13. Morkevicius, A., Jankevicius., N. “An approach: SysML-based automated requirements verification”, IEEE International Symposium on Systems Engineering (ISSE), pp. 92-97, Rome, 2015.
14. Object Management Group OMG Systems Modeling Language (OMG SysML), v1.5, May 2017. Available at https://www.omg.org/spec/SysML/1.5/.
15. Object Management Group OMG Unified Modeling Language (OMG UML), v2.5.1, December 2017. Available at https://www.omg.org/spec/UML/2.5.1/.
16. Soegaard, S. E. “Adopting MBSE using SysML in System Development–Joint Strike Missile (JSM)”, No Magic World Symposium, May 2016. Available at https://vimeopro.com/user28256466/nomagic-world-symposium-2016-presentations/ page/3.
17. Spangelo, S. C. et al. “Applying model based systems engineering (MBSE) to a standard CubeSat”, Aerospace Conference IEEE, 2012.
18. Walden, David, ed. INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, v.4. International Council on System Engineering (INCOSE), August 2015.
19. Weilkiens, T. Systems Engineering with SysML/UML: Modeling, Analysis, Design. Boston: MK/OMG Press, 2008.
20. Zachman, J. A. “A framework for information systems architecture”, IBM Syst. J., vol. 26, no. 3, pp. 276-292, 1987.

About the Authors

Aiste Aleksandraviciene

Aiste is an OMG® Certified Systems Modeling Professional (OCSMP). She has been with No Magic Europe for eight years, and currently holds the position of Solutions Architect.

Her expertise area is MBSE, with special focus on requirements management. She takes responsibility for educating No Magic clients on three pillars of MBSE: language (SysML), method (MagicGrid), and tool (No Magic software and integrations). While working with internationally-known clients, such as Renault-Nissan, Leica Geosystems, Bombardier Transportation, and Kongsberg Defence & Aerospace, she provides trainings, gives tool demonstrations, and participates in providing custom solutions. Aiste is also responsible for spreading the MBSE culture by organizing MBSE webinars, producing and maintaining the training material, and writing papers (alone and in collaboration with her colleagues). She is a speaker at multiple MBSE conferences and other public events.

Before joining the Solution Architects team, Aiste worked as the Documentation Manager and has over five years of experience as a technical writer. While producing technical documentation of No Magic products, she gained a lot of experience with writing technical texts, which helped her greatly with producing the ***MagicGrid BoK***.

Aiste holds a master's degree in Information Systems Engineering from Kaunas University of Technology (Lithuania).

Aurelijus Morkevicius, Ph.D.

Aurelijus is an OMG® Certified UML, Systems Modeling, and BPM Professional. Currently he is the Head of Solutions Department at No Magic Europe.

He has expertise with model-based systems and software engineering (mostly based on UML and SysML) and defense architectures (DoDAF, MODAF, NAF). Aurelijus is working with companies such as General Electric, Bombardier Transportation, Deutsche Bahn, ZF, Ford, BAE Systems, SIEMENS, and BMW. He is also a co-chairman and one of the leading architects for the current OMG UAF (previously known as UPDM) standards development group. He represents the No Magic company in INCOSE and NATO.

In addition, Aurelijus is actively involved in educational activities. He teaches an Enterprise Architecture course in Kaunas University of Technology; he earned his PhD in Informatics Engineering at the same university in 2013. Aurelijus is also an author of multiple articles, and a speaker at multiple conferences.

Aiste Aleksandraviciene
Aurelijus Morkevicius, Ph.D.

MagicGrid®
Book of Knowledge
A Practical Guide to Systems Modeling
using MagicGrid from No Magic

Language editor Nina K Pettis
Designed by Skaidra Vaicekauskiene
Cover Design David Fiegenschue, Fig Design

Published and printed by
Vitae Litera, UAB
Savanoriu av. 137
LT-44146 Kaunas, Lithuania
www.tuka.lt | info@tuka.lt