



Aistė Aleksandravičienė
Aurelijus Morkevičius, PH.D.

MagicGrid® BOOK OF KNOWLEDGE

A Practical Guide to Systems Modeling using
MagicGrid from Dassault Systèmes

2nd edition



Aistė Aleksandravičienė
Aurelijus Morkevičius, PH.D.

MagicGrid® BOOK OF KNOWLEDGE

**A Practical Guide to Systems Modeling using
MagicGrid from Dassault Systèmes**

2nd edition



Kaunas, 2021

ISBN 978-609-454-554-2

© 2018-2021, No Magic, Inc. All Rights reserved.

While No Magic, Inc. has taken precautions to ensure accuracy and completeness, the information contained in this book is provided for information purposes only and No Magic, Inc. does not make any representations as to its accuracy or completeness or any commitment as to future availability of functionality or products. No Magic, Inc. disclaims any liability, loss, or risk incurred as a result of the use of any information or advice contained in this book, either directly or indirectly. Individual results may vary. No part of this publication may be reproduced in any form or by any means without the prior written permission of No Magic, Inc. All other trademarks are the property of their respective owners.

TABLE OF CONTENTS

FOREWORD BY OLIVIER SAPPIN	7
FOREWORD BY AURELIJUS MORKEVIČIUS	8
CONTRIBUTORS	9
REVIEWERS	10
PREFACE	11
PROBLEM DOMAIN	15
Black-box perspective	15
Stakeholder Needs	16
Step 1. Organizing the model for stakeholder needs	17
Step 2. Creating a table for stakeholder needs	18
Step 3. Capturing stakeholder needs	19
Capturing information directly in the model	20
Copying information from a Microsoft Excel spreadsheet	20
Synchronizing information from a Microsoft Excel spreadsheet	22
Importing information from a ReqIF file	24
Synchronizing information from IBM Rational DOORS	26
Step 4. Grouping stakeholder needs	29
Step 5. Numbering stakeholder needs	31
Step 6. Categorizing stakeholder needs	33
Stakeholder Needs done. What's next?	34
System Context	35
Step 1. Organizing the model for the system context	36
Step 2. Capturing system contexts	37
Step 3. Creating an ibd for a system context	38
Step 4. Capturing participants of the system context	38
Step 5. Specifying interactions between the participants of system context	41
Step 6. Adding item flows to the system context	42
System Context done. What's next?	44
Use Cases	45
Step 1. Organizing the model for use cases	47
Step 2. Creating a diagram for use cases	47
Step 3. Capturing use cases	50
Step 4. Creating a diagram for specifying the use case scenario	50
Step 5. Creating swimlanes and setting allocation mode	51
Step 6. Specifying steps of the use case scenario	53
Step 7. Adding item flows to the use case scenario	58
Step 8. Supplementing the use case scenario with a parallel flow	61
Use Cases done. What's next?	62
Measures of Effectiveness	63
Step 1. Organizing the model for MoEs	64
Step 2. Creating a block for capturing MoEs	64
Step 3. Capturing MoEs for the Sol	65
MoEs done. What's next?	69

White-box perspective	70
Functional Analysis	71
Step 1. Organizing the model for functional analysis	73
Step 2. Creating an activity diagram to decompose a function	73
Step 3. Specifying the white-box scenario	76
Step 4. Creating an activity decomposition map	78
Functional Analysis done. What's next?	79
Conceptual Subsystems	80
Step 1. Organizing the model for conceptual subsystems communication	83
Step 2. Creating a bdd for specifying conceptual interfaces	83
Step 3. Capturing conceptual interfaces	85
Step 4. Creating an ibd for capturing conceptual subsystems	87
Step 5. Capturing conceptual subsystems	89
Step 6. Specifying interactions between conceptual subsystems and the outside of the Sol	90
Step 7. Specifying interactions among conceptual subsystems	94
Step 8. Allocating functions by conceptual subsystems and synchronizing item flows	99
Conceptual Subsystems done. What's next?	105
MoEs for Subsystems	106
Step 1. Organizing the model for MoEs for subsystems	106
Step 2. Capturing MoEs for subsystems	107
MoEs for Subsystems done. What's next?	110
Traceability to Stakeholder Needs	111
Step 1. Organizing the model for traceability	113
Step 2. Creating a matrix for capturing refine relationships	113
Step 3. Capturing refine relationships	115
Problem Domain done. What's next?	117
SOLUTION DOMAIN	118
Building the logical architecture of the Sol	121
System Requirements	122
Step 1. Creating and organizing a model for system requirements	124
Step 2. Creating a diagram for system requirements	126
Step 3. Specifying system requirements	126
Step 4. Establishing traceability to stakeholder needs	128
Step 5. Establishing traceability to the rest of the problem domain model	131
System Requirements done. What's next?	133
System Structure	134
Step 1. Organizing the model for the LSA	136
Step 2. Creating a bdd for capturing the LSA	137
Step 3. Capturing logical subsystems	137
Step 4. Establishing traceability to the problem domain model	140
Step 5. Capturing logical interfaces	142
System Structure done. What's next?	149
System Behavior	150
System Parameters	152
Step 1. Organizing the model for system parameters	154
Step 2. Specifying the MoE for capturing the total energy consumption	155
Step 3. Defining a method for calculating the total energy consumption	158
Step 4. Specifying system parameters for calculating the total energy consumption	160
Step 5. Binding constraint parameters to corresponding value properties	162
Step 6. Performing early system requirements verification	166
Step 7. Storing values in the model	172
System Parameters done. What's next?	176

Traceability to System Requirements	177
Step 1. Creating a matrix for capturing satisfy relationships	178
Step 2. Capturing satisfy relationships	179
LSA done. What's next?	180
Building the logical architecture of subsystems	181
Subsystem Requirements	182
Step 1. Creating and organizing a model for subsystem requirements	184
Step 2. Creating a diagram for subsystem requirements	186
Step 3. Specifying subsystem requirements	186
Step 4. Specifying traceability relationships	188
Subsystem Requirements done. What's next?	193
Subsystem Structure	194
Step 1. Organizing the model for subsystem structure	195
Step 2. Getting ready to capture the subsystem structure	196
Step 3. Capturing components of the Cooling System	200
Step 4. Creating an ibd for specifying interactions	202
Step 5. Specifying interactions with the outside	204
Step 6. Specifying interactions within the Cooling System	209
Subsystem Structure done. What's next?	214
Subsystem Behavior	215
Step 1. Organizing the model for subsystem behavior	217
Step 2. Creating a diagram for capturing Cooling System states	217
Step 3. Capturing states of the Cooling System	218
Step 4. Specifying event occurrences on transitions	220
Step 5. Specifying internal behaviors of the state	225
Step 6. Synchronizing item flows from the structure model	240
Subsystem behavior done. What's next?	242
Subsystem Parameters	243
Step 1. Organizing the model for subsystem parameters	245
Step 2. Specifying the MoE for capturing the total energy consumption for cooling	246
Step 3. Defining a method to calculate the total energy consumption for cooling	248
Step 4. Specifying subsystem parameters for calculating the total energy consumption for cooling	249
Step 5. Binding constraint parameters to appropriate value properties	249
Step 6. Performing early subsystem requirements verification	251
Step 7. Recalculating subsystem parameters in different states	252
Subsystem parameters done. What's next?	255
Traceability to Subsystem Requirements	256
Step 1. Creating a dependency matrix for capturing satisfy relationships	257
Step 2. Capturing satisfy relationships to subsystem requirements	258
Step 3. Traceability to and revision of system requirements	260
LSSA done. What's next?	263
Building system configuration model	265
System Configuration Structure	266
Step 1. Creating and organizing a model for system configuration structure	268
Step 2. Getting ready to capture the structure of the system configuration	269
Step 3. Capturing the integrated structure of the system configuration	273
Step 4. Verifying interface compatibility	277
System Configuration Structure done. What's next?	282
System Configuration Behavior	283
Step 1. Starting simulation of the system configuration block	284
Step 2. Analyzing the integrated behavior of the system configuration	285
System Configuration Behavior done. What's next?	289

System Configuration Parameters	290
Step 1. Redefining the MoE	291
Step 2. Creating a parametric diagram	291
Step 3. Updating bindings	293
System Configuration Parameters done. What's next?	295
Traceability from System Configuration to System Requirements	296
Solution Domain done. What's next?	297
IMPLEMENTATION DOMAIN	298
Implementation Requirements	298
ANNEX A: SAFETY & RELIABILITY ANALYSIS	301
Stakeholder needs: safety and reliability requirements	302
Conceptual FMEA in Black Box view	302
Functional FMEA in Black Box view	304
Functional FMEA coverage analysis	306
Addressing safety and reliability requirements in Problem domain	307
FMEA at the White Box view	311
Addressing safety and reliability concerns in the LSA	312
Addressing safety and reliability concerns at the subsystem level	315
Cooling Subsystem level	315
Filtering Subsystem level	317
Logical subsystems FMEA at the subsystem level	328
Addressing safety and reliability concerns at the configuration level	329
ANNEX B: FROM SYSTEM OF SYSTEMS TO SYSTEM ARCHITECTURE	330
System of Systems Engineering	330
Transitioning from SoS to SA	331
AFTERWORD	335
GLOSSARY	336
BIBLIOGRAPHY	344
ABOUT THE AUTHORS	345

FOREWORD BY OLIVIER SAPPIN

Model-based systems engineering (MBSE) is a relatively young practice for developing complex systems. The promise of MBSE is to be able to progressively model and simulate systems behavior in accordance with the needs & scenarios of usage, at any time. This promise is about avoiding the discipline-specific tunnel effect and giving project stakeholders the ability to continuously monitor the evolution of system development. One of the key success factors of MBSE is to apply best practices based on a sound systems engineering methodology.

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 CATIA® 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 Dassault Systèmes name behind this effort.

Thank you to all the magicians who made this happen, especially our industry process consultants 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.

— Olivier Sappin, CATIA CEO
Dassault Systèmes

FOREWORD BY AURELIJUS MORKEVIČIUS

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 the team, 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 and colleagues, 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 MBSE Cyber Experience 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 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-by-step 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 Morkevičius, PhD
Dassault Systèmes

CONTRIBUTORS

Andrius Armonas, PhD, andrius.armonas@3ds.com

Barry Papke, barry.papke@3ds.com

Daniel Brookshier, daniel.brookshier@3ds.com

Edita Milevičienė, edita.mileviciene@3ds.com

Elona Paulikaitytė, elona.paulikaityte@3ds.com

Gintarė Kriščiūnienė, gintare.krisciuniene@3ds.com

Osvaldas Jankauskas, osvaldas.jankauskas@3ds.com

Rokas Bartkevičius, rokas.bartkevicius@3ds.com

Ronald Kratzke, ronald.kratzke@3ds.com

Saulius Pavalkis, PhD, saulius.pavalkis@3ds.com

Solange Muhayimana, solange.muhayimana@3ds.com

Thomas Marchand, thomas.marchand@3ds.com

Tomas Juknevičius, tomas.juknevicius@3ds.com

Žilvinas Strolia, zilvinas.strolia@3ds.com

REVIEWERS

Andrew Terbrock
Antonio Tulelli
Arnaud Durantin
Atif Mahboob
Bernhard Meyer
Bryan K. Pflug
Carmelo Tommasi
Kris Gough
Christian Guist
Christian Konrad
Damian Delic
Darius Šilingas
David D. Walden
David Hughes
Jeff Tipps
Jon Symons
Gauthier Fanmuy
Himanshu Upadhyay
Holger Gamradt
James Towers
Jonathan Jansen
Macaulay Osaisai
Michael Schneider
Michael Elliot
Oliver Nagel
Olivier Casse
Paolo Neri
Pierfelice Ciancia
Piotr Malecki
Prem Sharma
Raymond Kempkens
Robert Marino
Sagar Nirgudkar
Shelley Higgins
Sven Degenkolbe
Thomas Eickhoff
Timo Wekerle
Toshihiro Obata
Ulf Koenemann
Zackary Bogner

PREFACE

Why this book?

Any theory needs practice. An approach becomes useful when it is provided together with comprehensive instructions on how to use it in practice. The MagicGrid framework for **Model-Based Systems Engineering (MBSE)** by CATIA® No Magic is no exception to this rule. This was the main reason we have decided to write this book.

So, if you're new at **MBSE** or look forward to learn about the MagicGrid framework from A to Z, this book is for you. This book is for you, too, if you wish to learn how to use CATIA Magic software for **MBSE** in combination with **OMG® Systems Modeling Language (SysML)™** and MagicGrid. This book is for you as well, 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 CATIA Magic software for MBSE; that is, Magic Cyber-Systems Engineer / Cameo Systems Modeler or Magic Systems of Systems Architect / Cameo Enterprise Architecture along with the Cameo DataHub plugin installed on the top, or Magic Software Architect / MagicDraw with the SysML and Cameo DataHub plugins, as of version 19.0, generally referred to as the **modeling tool**.

Why MagicGrid?

MagicGrid is the bridge between the theory and practice, bringing three major MBSE components – method, language, and modeling tool – together.

The idea of developing a new framework 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 critical enabler 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 framework 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 framework is completely applicable in practice for the following reasons:

- It is fully compatible with "vanilla" SysML, which means, no extensions for the standard 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 framework can be represented as a Zachman style matrix (see the following figure), 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.

	Pillar						
Domain			Requirements	Structure	Behavior	Parameters	Safety & Reliability
	Problem	Black Box	Stakeholder Needs	System Context	Use Cases	Measures of Effectiveness (MoEs)	Conceptual and Functional Failure Mode & Effects Analysis (FMEA)
		White Box		Conceptual Subsystems	Functional Analysis	MoEs for Subsystems	Conceptual Subsystems FMEA
	Solution	System Requirements	System Structure	System Behavior	System Parameters	System Safety & Reliability (S&R)	
		Subsystem Requirements	Subsystem Structure	Subsystem Behavior	Subsystem Parameters	Subsystem S&R	
		Component Requirements	Component Structure	Component Behavior	Component Parameters	Component S&R	
	Implementation	Implementation Requirements					

As you can see, the framework defines the Problem, Solution, and Implementation domains for developing the **system of interest (Sol)**. 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 **Sol** 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 implementation requirements specification in this domain is not a part of MBSE and therefore appears outside the scope of the matrix.

Each domain definition includes different aspects of the **Sol**. These aspects match the four pillars of the SysML, that is, Requirements, Structure, Behavior, and Parameters (also known as Parametrics); plus, the Safety & Reliability aspect added in the 2nd edition of this book. They are represented as columns of the matrix.

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, including the elements that can be used for modeling there, 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 **Sol** implementation.

Alignment to ISO/IEC/IEEE 15288

Before we explore MagicGrid in detail, it is important to disclose how it fits into the overall systems engineering life cycle defined by ISO/IEC/IEEE 15288.

MagicGrid actually addresses the technical ISO/IEC/IEEE 15288 processes only.

To start with the *Business or Mission Analysis* process, it is considered to be the input to the Problem domain of the MagicGrid and is usually described by using other technologies, like **OMG® Unified Architecture Framework (UAF)®**, which can be utilized in the same **modeling tool**, prior to applying the MagicGrid framework for developing the identified systems.

The Problem domain of this framework covers the entire *Stakeholder Needs and Requirements Definition* process. The subprocesses are mapped as follows:

- *Stakeholder Needs Definition* subprocess to the **Stakeholder Needs** cell of MagicGrid.
- *Operational Concept and Other Life Cycle Concepts Development* subprocess to the **System Context**, **Use Cases**, and **Measures of Effectiveness (MoEs)** cells, all together.
- *Transformation of Stakeholder Needs into Stakeholder Requirements* subprocess to the **Conceptual Subsystems**, **Functional Analysis**, and **MoEs for Subsystems** cells, all together.

The *System Requirements Definition* process is carried out in the **System Requirements** cell, although it can be extended in the **Subsystem Requirements** and Component Requirements cells, too.

Right after system requirements are completed, the *Architecture Definition* process takes place. In MagicGrid, it covers all system-level and subsystem-level cells of the **Solution domain**, except the **System Requirements** and **Subsystem Requirements** cells. The *Architecture Definition* process also covers the component-level cells, but only partially. These cells are covered by the *Design Definition* process, too. This is where the architecture relates to design, or in other words, **MBSE** meets **Model-Based Design (MBD)**, and activities of the *Assess architecture candidates* subprocess are performed. Once they are completed, the *Design Definition* process is carried out.

Finally, the *Implementation process* transforms requirements, architecture, and design into actions that create the system element according to the practices of the selected implementation technology, using appropriate technical specialties or disciplines. It is covered in the **Implementation domain** of MagicGrid.

		Pillar			
Domain	Problem	Requirements	Structure	Behavior	Parameters
		Stakeholder Needs Definition	Operational Concept and Other Life Cycle Concepts Development		
	Solution	Transformation of Stakeholder Needs into Stakeholder Requirements			
		System Requirements Definition	Architecture Definition		Design Definition
	Implementation	Implementation Requirements Definition	Implementation		

ISO 15288

Case study

In this book, the modeling framework applies to the specification and architecture 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

It's important to understand that this book 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** (including sub-chapters that correspond different phases of creating the solution domain model), and the **Implementation domain**.

While chapters consist of sub-chapters, the latter consist of multiple sections, each named after relevant cell of the MagicGrid framework. The order of these cells defines the modeling workflow. Every section includes the brief overview of the cell to explain the purpose of modeling in this cell, who is responsible for the results, and how to model. It also gives the list of basic steps for the modeler to go through to achieve the expected result. Each step is described in the related sub-section, which 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. If the version of your **modeling tool** is 2021x or later, you are welcome to study the sample model there; it is saved as *VCCS (MG BoK v2 Sample).mdzip*.

There are information boxes marked with the  symbol, in the text. These boxes are designated to provide you with additional information, tips and tricks for smarter modeling, or references to more detailed information sources.

Moreover, the second edition of the book includes:

- **Annex A**, which provides comprehensive instructions on how to enrich the system model with safety and reliability information.
- **Annex B**, which provides a detailed description of transitioning from System of Systems to a single system architecture in the model-based environment.

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 [NoMagic.MagicGrid@3ds.com](mailto>NoMagic.MagicGrid@3ds.com).

PROBLEM DOMAIN

The purpose of the problem domain definition is to analyze stakeholder needs and refine them with SysML model elements to get the clear and coherent description of what functions are expected from the **Sol** in the defined environment.

As shown in the following figure, the problem domain analysis is performed in two phases. In the first phase, the **Sol** is considered a black box. The main focus is on how it interacts with the defined environment without getting any knowledge about its internal structure and behavior (i.e., the operational analysis of the **Sol** is performed). In the second phase, the black box is opened and the **Sol** is analyzed from the white-box perspective, which helps to understand the expected behavior and conceptual structure of the **Sol** (i.e., the functional analysis of the **Sol** is performed). It is also important to note that the problem definition is not about the logical or the physical architecture of the **Sol**.

As you can see in the following figure, both phases of the problem domain definition consist of specifying the requirements, structure, behavior, and parameters of the **Sol**. The only difference is in perspective.

		Pillar				
Domain	Problem	Requirements	Structure	Behavior	Parameters	Safety & Reliability
		Stakeholder Needs	System Context	Use Cases	Measures of Effectiveness (MoEs)	Conceptual and Functional Failure Mode & Effects Analysis (FMEA)
		Conceptual Subsystems	Functional Analysis	MoEs for Subsystems	Conceptual Subsystems FMEA	
	Solution	System Requirements	System Structure	System Behavior	System Parameters	System Safety & Reliability (S&R)
		Subsystem Requirements	Subsystem Structure	Subsystem Behavior	Subsystem Parameters	Subsystem S&R
	Implementation	Component Requirements	Component Structure	Component Behavior	Component Parameters	Component S&R
	Implementation	Implementation Requirements				

Black-box perspective

Problem domain analysis and definition from the black-box perspective is the first step towards specifying what the **Sol** shall do fulfill the stakeholders' expectations. The goal of this phase is to understand how the **Sol** interacts with its environment to solve the given challenges. As the **Sol** is considered a black box, the analysis is not concerned with its internal structure and behavior. The analysis is performed to define the main inputs and outputs, black-box functions, and quantifiable characteristics of the **Sol**, while operating in a variety of system contexts.

The following pages describe how to capture the problem domain definition from the black-box perspective in the SysML model.

		Pillar				
Domain	Problem	Requirements	Structure	Behavior	Parameters	Safety & Reliability
		Stakeholder Needs	System Context	Use Cases	Measures of Effectiveness (MoEs)	Conceptual and Functional Failure Mode & Effects Analysis (FMEA)
			Conceptual Subsystems	Functional Analysis	MoEs for Subsystems	Conceptual Subsystems FMEA
	Solution	System Requirements	System Structure	System Behavior	System Parameters	System Safety & Reliability (S&R)
		Subsystem Requirements	Subsystem Structure	Subsystem Behavior	Subsystem Parameters	Subsystem S&R
		Component Requirements	Component Structure	Component Behavior	Component Parameters	Component S&R
	Implementation	Implementation Requirements				

Stakeholder Needs

What is it?

The cell represents information gathered from various **stakeholders** of the **Sol**. This information includes primary user needs, system-related government regulations, industry standards, policies, procedures, and internal guidelines, etc.

Stakeholder needs can be captured by interviewing **stakeholders**, giving them questionnaires, discussing needs in focus groups, or studying documents written in diverse formats. Though this 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. Once this is done, traceability relationships are created to convey which artifacts from the problem domain model refine every stakeholder need (see Chapter **Traceability to Stakeholder Needs**). These refinements will serve as a basis for specifying system requirements (see Chapter **System Requirements**).

Who is responsible?

The elicitation and analysis of stakeholder needs can be performed by the Requirements Engineer or Business and Mission Analyst.

How to model?

Stakeholder needs can be captured directly in the [modeling tool](#), or imported from other tools and formats, including:

- Requirements management tools (i.e., Requirements Engineer (TRM) (as of 19.0 SP4), IBM® Rational® DOORS®)
- Spreadsheets (i.e., Microsoft Excel file)
- ReqIF file (which contains data exported from another tool, such as, 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 and have a unique identification number, 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 source, for example). It can be two categories of them: functional and non-functional. This facilitates further analysis of stakeholder needs.

#	△ Name	Text
1	□ R SN-1 Stakeholder Needs	
2	□ R SN-1.1 User Needs	
3	R SN-1.1.1 Sound Level	Climate control unit in max mode shall not be louder than engine.
4	F SN-1.1.2 Manual Control	I should be able to start and stop climate control by myself.
5	F SN-1.1.3 Heating & Cooling	The unit must be able to heat and cool.
6	R SN-1.1.4 Energy Consumption	I'd like the unit to consume as little energy as possible.
7	F SN-1.1.5 Ambient Temperature	I want to see the ambient temperature on the screen or some other output device.
8	F SN-1.1.6 Desired Temperature	It should be a possibility to easily specify the desired temperature.
9	F SN-1.1.7 Comfortable Temperature	I'd like to feel comfortable temperature while being in the cabin
10	□ R SN-1.2 Design Constraints	
11	R SN-1.2.1 Total Mass	Mass of the unit shall not exceed 2 percent of the total car mass.

Tutorial

- [Step 1. Organizing the model for stakeholder needs](#)
- [Step 2. Creating a table for stakeholder needs](#)
- [Step 3. Capturing stakeholder needs](#)
- [Step 4. Grouping stakeholder needs](#)
- [Step 5. Numbering stakeholder needs](#)
- [Step 6. Categorizing stakeholder needs](#)

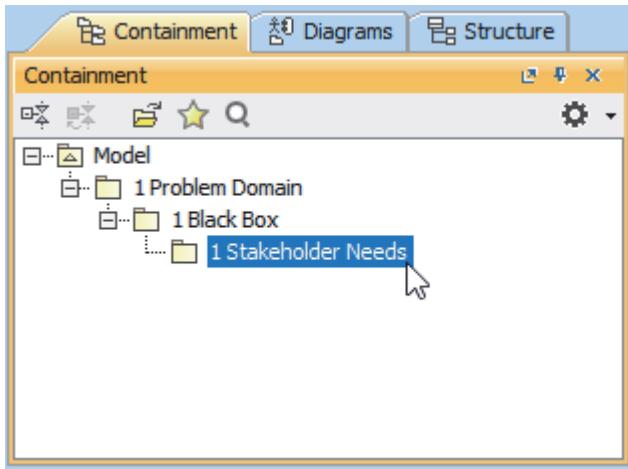
Step 1. Organizing the model for stakeholder needs

A well-organized model is easier to read, understand, and maintain; thus, we recommend organizing your model into packages. Packages in a SysML model are very similar to folders in a file system: while folders are used to organize files (including other folders), 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, as 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.

- i** You can skip this step (as well as other steps of establishing the model structure), if you use the *MagicGrid v2 QuickStart* or *MagicGrid v2 Blank* template. They both come with the 2021x version of the [modeling tool](#), and include a predefined structure based on the MagicGrid framework.



- i** Using numbers in package names enables you to preserve the package order in the model tree.

To organize the model for stakeholder needs

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

There are many ways to capture stakeholder needs in the SysML model. You can do this directly in the Model Browser or over a presentation view (i.e., a requirement diagram or table), or another diagram, which can display requirements (for example, [block definition diagram \(bdd\)](#)). In this tutorial, we utilize the requirement table, as it is the best way to get stakeholder needs into the model from an external source, such as Microsoft Excel spreadsheet or IBM® Rational® DOORS®.

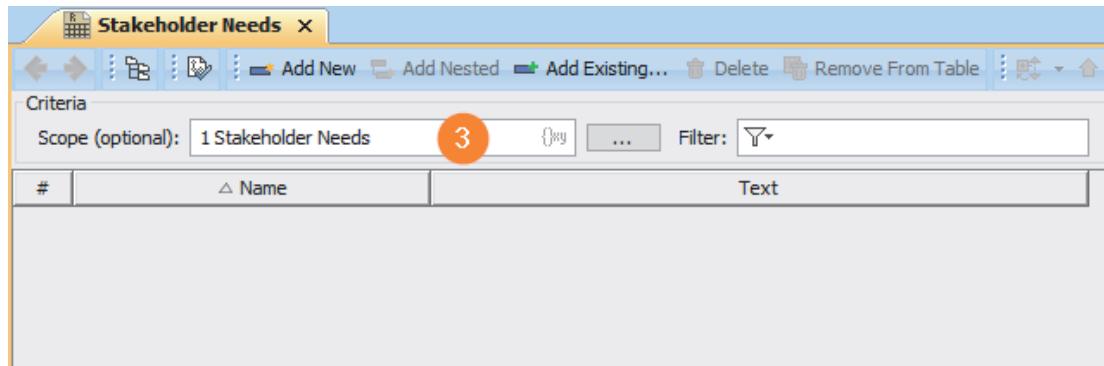
It is important to notice that the MagicGrid BoK always describes one possible way to complete the task (for example, we choose to create requirement table to capture and display stakeholder needs for the reason explained above), but this does not mean that you cannot do this in another way. For a comprehensive description of all the alternatives, refer to the latest documentation of the [modeling tool](#).

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.

i Note that the table is named after the package where it is 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 related package.



Step 3. Capturing stakeholder needs

Suppose you need to capture the following stakeholder needs into your model:

#	Name	Text
1	Ambient Temperature	I want to see the ambient temperature on the screen or some other output device.
2	Comfortable Temperature	I'd like to feel comfortable temperature while being in the cabin.
3	Desired Temperature	It should be a possibility to easily specify the desired temperature.
4	Heating & Cooling	The unit must be able to heat and cool.
5	Energy Consumption	I'd like the unit to consume as little energy as possible.
6	Manual Control	I should be able to start and stop climate control by myself.
7	Sound Level	Climate control unit in max mode shall not be louder than engine.
8	Total 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 a unique identification, name, and text specification.

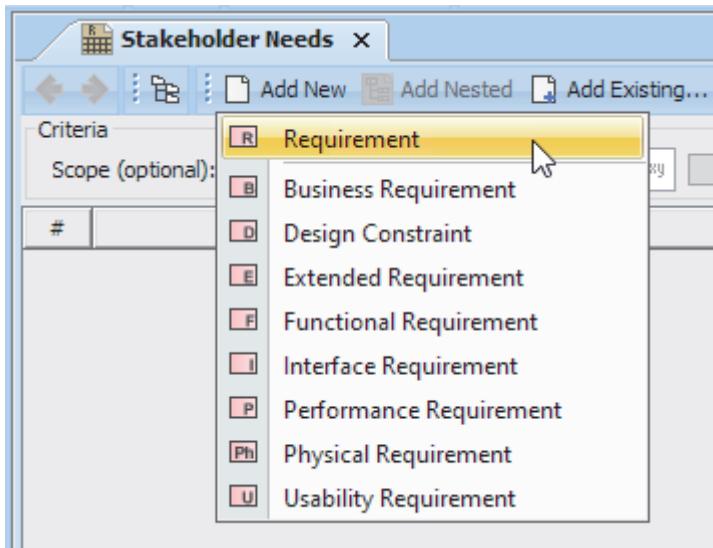
You can get the stakeholder needs into your model by one of the following means:

- Capturing information directly in the model
- Copying information from a Microsoft Excel spreadsheet
- Synchronizing 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 this cell tutorial](#), if not already 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, in [step 3 of this cell tutorial](#).
4. Repeat steps 2 and 3 until you have all items in the table, as displayed in the following figure.

A screenshot showing two parts of a modeling tool. On the left is the "Containment" view of the Model Browser, which shows a hierarchy: Model > 1 Problem Domain > 1 Block Box > 1 Stakeholder Needs. This node is selected. On the right is a "Requirements" table titled "1 Stakeholder Needs". The table has columns for "#", "Name", and "Text". It contains eight rows, each representing a requirement with a description. For example, requirement 1 is "Ambient Temperature" with the text "I want to see the ambient temperature on the screen or some other output device." Requirements 2 through 8 are also listed with their respective descriptions.

i As you can see, items created in the SysML requirements table also appear in the Model Browser.

Copying information from a Microsoft Excel spreadsheet

If you are provided with a 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	Title	Description
2	Ambient Temperature	I want to see the ambient temperature on the screen or some other output device.
3	Comfortable Temperature	I'd like to feel comfortable temperature while being in the cabin.
4	Desired Temperature	It should be a possibility to easily specify the desired temperature.
5	Energy Consumption	I'd like the unit to consume as little energy as possible.
6	Heating & Cooling	The unit must be able to heat and cool.
7	Manual Control	I should be able to start and stop climate control by myself.
8	Sound Level	Climate control unit in max mode shall not be louder than engine.
9	Total Mass	Mass of the unit shall not exceed 2 percent of the total car mass.

2. Copy the contents:

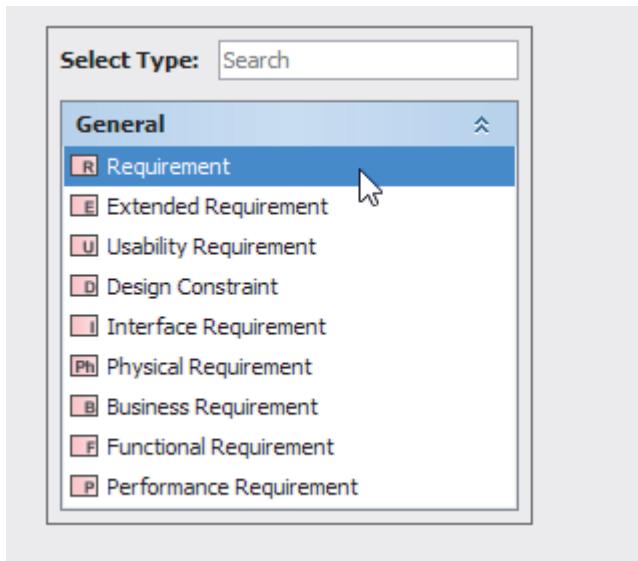
- Press Ctrl+C on Windows or Linux.
- Press Command+C on macOS.

3. Switch to the **modeling tool** and open the SysML requirement table created in [step 2 of this cell tutorial](#), if not already 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.

The screenshot shows the MagicGrid interface. On the left, the Model Browser displays a tree structure under 'Model' with '1 Problem Domain' containing '1 Black Box' and '1 Stakeholder Needs'. Under '1 Stakeholder Needs', there are eight items: '1 Total Mass', '2 Sound Level', '3 Manual Control', '4 Heating & Cooling', '5 Energy Consumption', '6 Ambient Temperature', '7 Desired Temperature', and '8 Comfortable Temperature'. To the right, a table titled 'Stakeholder Needs' is shown with the following data:

#	Name	Text
1	1 Total Mass	Mass of the unit shall not exceed 2 percent of the total car mass.
2	2 Sound Level	Climate control unit in max mode shall not be louder than engine.
3	3 Manual Control	I should be able to start and stop climate control by myself.
4	4 Heating & Cooling	The unit must be able to heat and cool.
5	5 Energy Consumption	I'd like the unit to consume as little energy as possible.
6	6 Ambient Temperature	I want to see the ambient temperature on the screen or some other output device.
7	7 Desired Temperature	It should be a possibility to easily specify the desired temperature.
8	8 Comfortable Temperature	I'd like to feel comfortable temperature while being in the cabin.

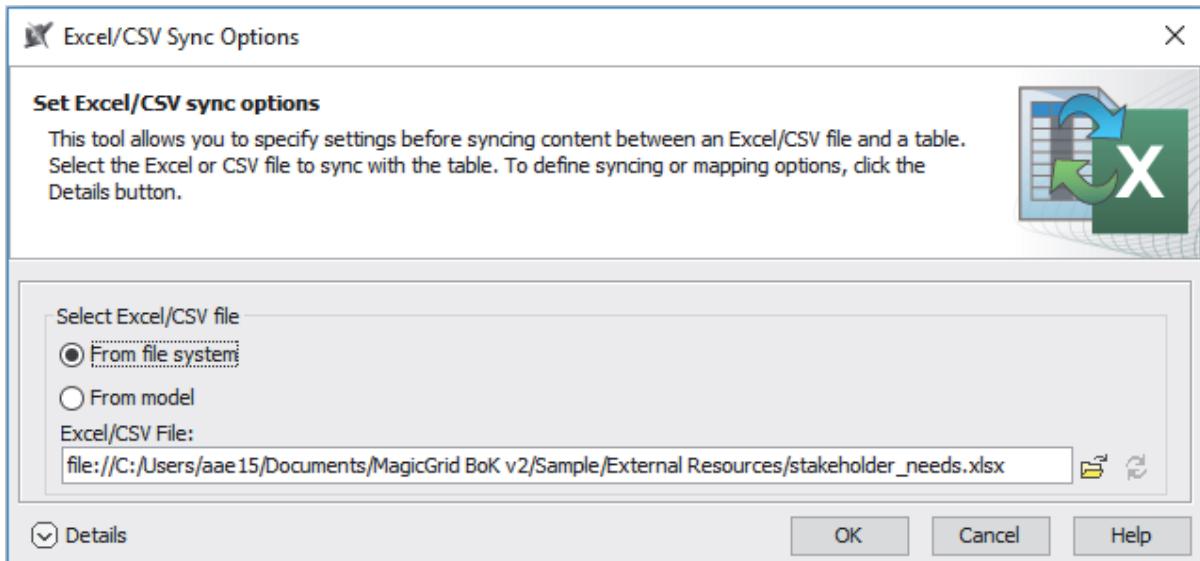
i As you can see, items copied to the SysML requirements table also appear in the Model Browser.

Synchronizing information from a Microsoft Excel spreadsheet

Another option to bring stakeholder needs from an MS Excel spreadsheet into a SysML model is to synchronize the model with the spreadsheet.

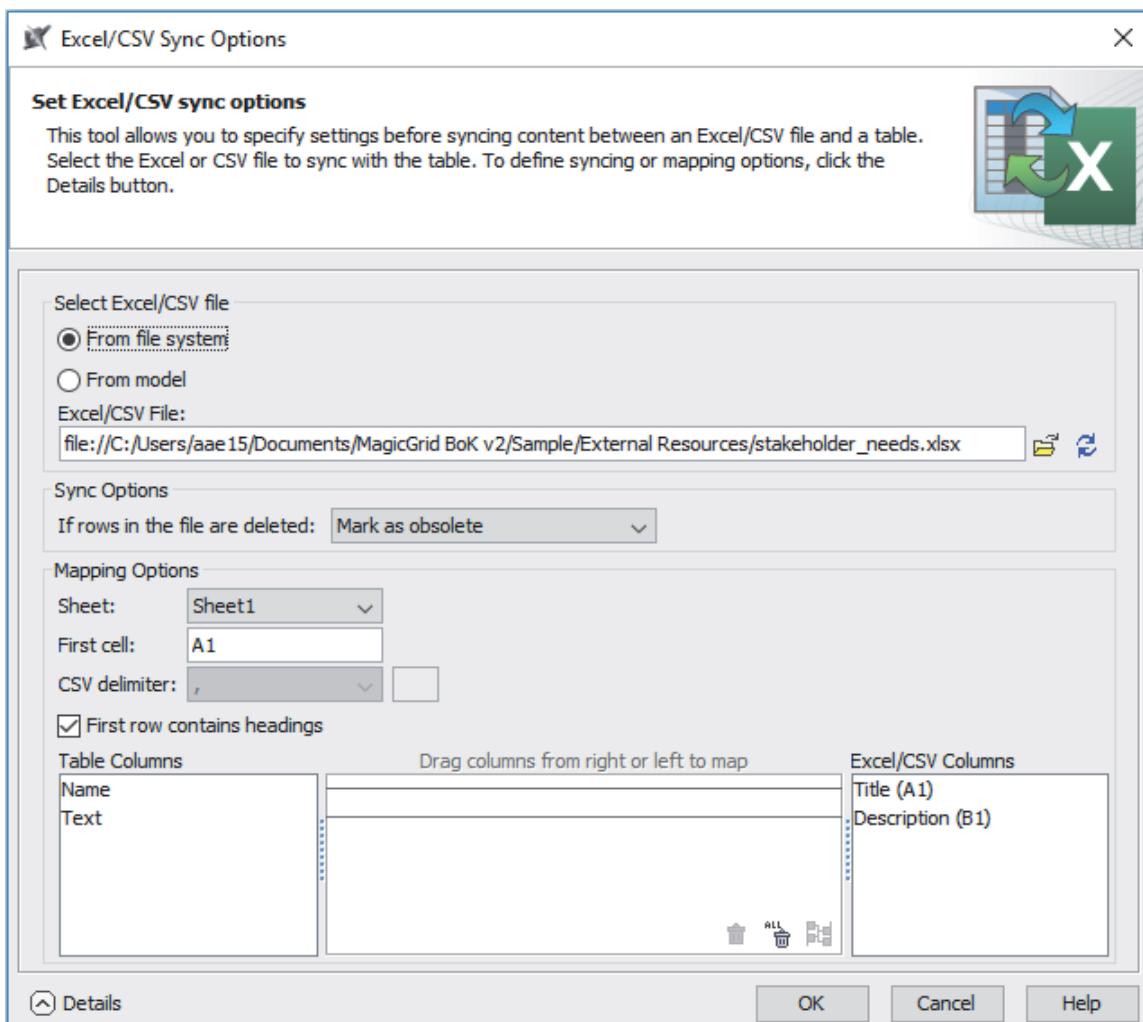
To synchronize stakeholder needs from the MS Excel spreadsheet

1. Open the table created in [step 2 of this cell tutorial](#), if not already opened.
2. On the toolbar of the table, click and then choose **Excel/CSV File > Select File**.
3. In the **Excel/CSV Sync Options** dialog, select the MS Excel file from your file system.



4. Click **Details** to expand the dialog.

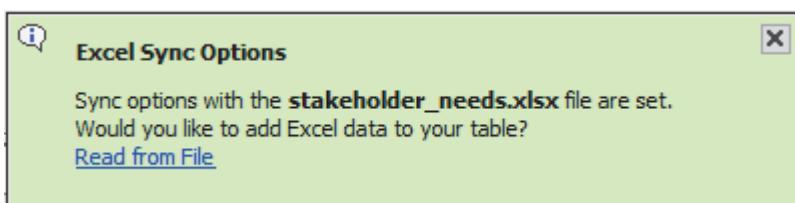
- Map columns of the SysML requirement table to columns of the MS Excel spreadsheet. For this, do the following:
 - Select a column on the left and drag it to the relevant column on the right.
 - Release the mouse.



- i** The modeling tool might require you to refresh the dialog before setting the mappings. Therefore, if you see this notification at the bottom of the dialog, click **Refresh**.

The mapping area is outdated. [Refresh?](#)

- Click **OK**.
- When the notification appears on the bottom right corner of the modeling tool window (see the following figure), click **Read from File**.



- Wait while stakeholder needs are synchronized with the table. The relevant notification displays in the bottom right corner, when synchronization is completed.

#	Name	Text
1	1 Total Mass	Mass of the unit shall not exceed 2 percent of the total car mass.
2	2 Sound Level	Climate control unit in max mode shall not be louder than engine.
3	3 Manual Control	I should be able to start and stop climate control by myself.
4	4 Heating & Cooling	The unit must be able to heat and cool.
5	5 Energy Consumption	I'd like the unit to consume as little energy as possible.
6	6 Ambient Temperature	I want to see the ambient temperature on the screen or some other output device.
7	7 Desired Temperature	It should be a possibility to easily specify the desired temperature.
8	8 Comfortable Temperature	I'd like to feel comfortable temperature while being in the cabin.

i Items synchronized to the SysML requirements table also appear in the Model Browser.

As you can see, all the items in the table are highlighted in green. The legend above reveals that all new items are highlighted like this. You can change the legend colors or hide the legend from the table. This, along with more information about using legends, can be found in the documentation of the [modeling tool](#).

Importing information from a ReqIF file

At times, stakeholder needs are gathered by using one of these tools:

- IBM® Rational® DOORS®
- PTC Integrity Modeler
- Polarion® REQUIREMENTS™
- Siemens Teamcenter
- Requirements Engineer (TRM)

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

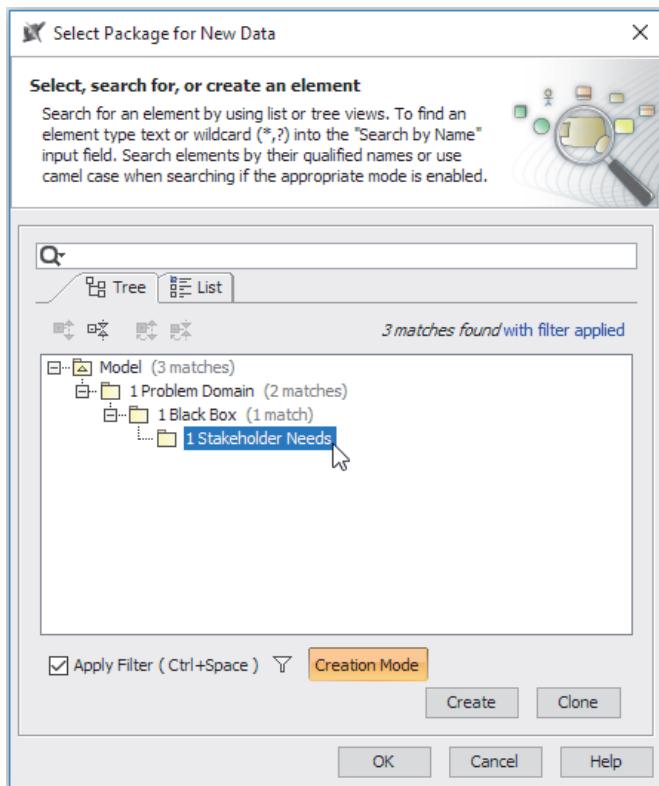
- Exporting requirements from the requirements management tool to a ReqIF file.
- Importing the ReqIF file to the SysML model in the [modeling tool](#).



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 from your file system and click **Open**.
3. In the **Select Package for New Data** dialog, select the package where you want to store the imported stakeholder needs (i.e., 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 this cell tutorial), all imported items also appear in the table.

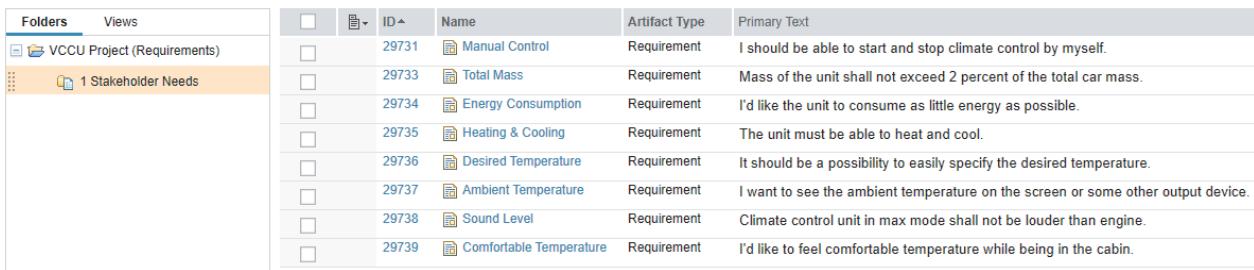
The screenshot shows two windows side-by-side. On the left is the 'Containment' view of the Model Browser, showing a tree structure with 'Model' expanded to show '1 Problem Domain', '1 Black Box', and '1 Stakeholder Needs'. The '1 Stakeholder Needs' node is selected and highlighted with a blue border. On the right is a table titled 'Stakeholder Needs' with 8 rows. The table has columns for '#', 'Name', and 'Text'. The data is as follows:

#	Name	Text
1	Ambient Temperature	I want to see the ambient temperature on the screen or some other output device.
2	Comfortable Temperature	I'd like to feel comfortable temperature while being in the cabin.
3	Desired Temperature	It should be a possibility to easily specify the desired temperature.
4	Energy Consumption	I'd like the unit to consume as little energy as possible.
5	Heating & Cooling	The unit must be able to heat and cool.
6	Manual Control	I should be able to start and stop climate control by myself.
7	Sound Level	Climate control unit in max mode shall not be louder than engine.
8	Total Mass	Mass of the unit shall not exceed 2 percent of the total car mass.

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](#).

- i** The [modeling tool](#) can be synchronized with DOORS only if it has the Cameo DataHub Plugin installed.



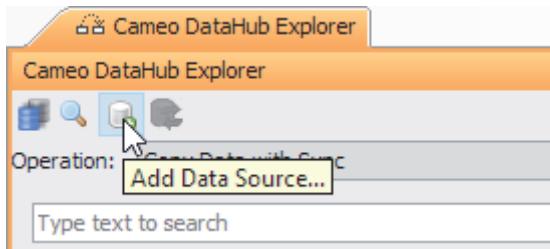
Folders	Views	ID	Name	Artifact Type	Primary Text
VCCU Project (Requirements)		29731	Manual Control	Requirement	I should be able to start and stop climate control by myself.
1 Stakeholder Needs		29733	Total Mass	Requirement	Mass of the unit shall not exceed 2 percent of the total car mass.
		29734	Energy Consumption	Requirement	I'd like the unit to consume as little energy as possible.
		29735	Heating & Cooling	Requirement	The unit must be able to heat and cool.
		29736	Desired Temperature	Requirement	It should be a possibility to easily specify the desired temperature.
		29737	Ambient Temperature	Requirement	I want to see the ambient temperature on the screen or some other output device.
		29738	Sound Level	Requirement	Climate control unit in max mode shall not be louder than engine.
		29739	Comfortable Temperature	Requirement	I'd like to feel comfortable temperature while being in the cabin.

In this tutorial, we assume that DOORS are used only for capturing stakeholder needs, and further analysis is performed in the SysML model. For this reason, we need to establish one-way 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.

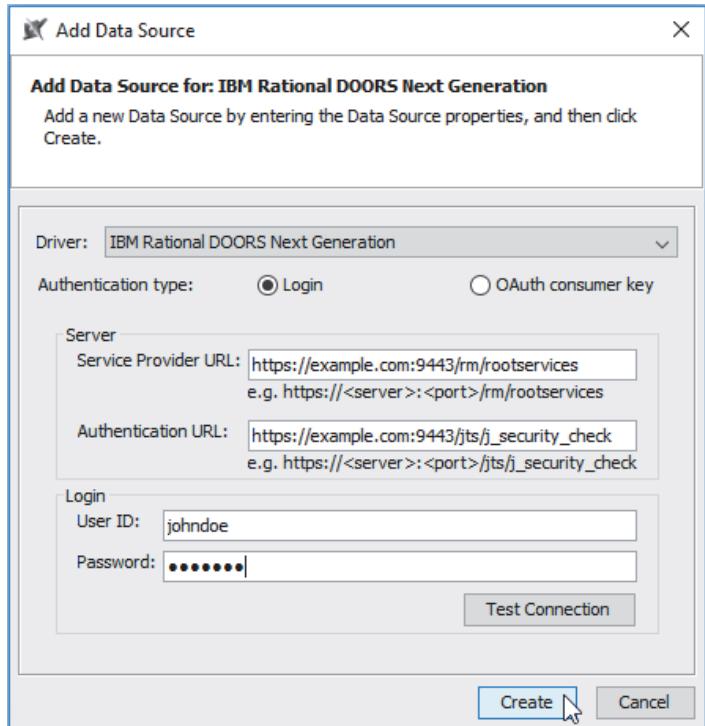
- i** Although the following procedure defines the steps for synchronization with IBM® Rational® DOORS® Next Generation 6.x, it is also possible to synchronize requirements from IBM® Rational® DOORS® 8.0-8.3 and 9.0-9.6 as well as from IBM® Rational® DOORS® NG 4.x and 5.x.

To synchronize stakeholder needs from DOORS NG 6.x to the [modeling tool](#)

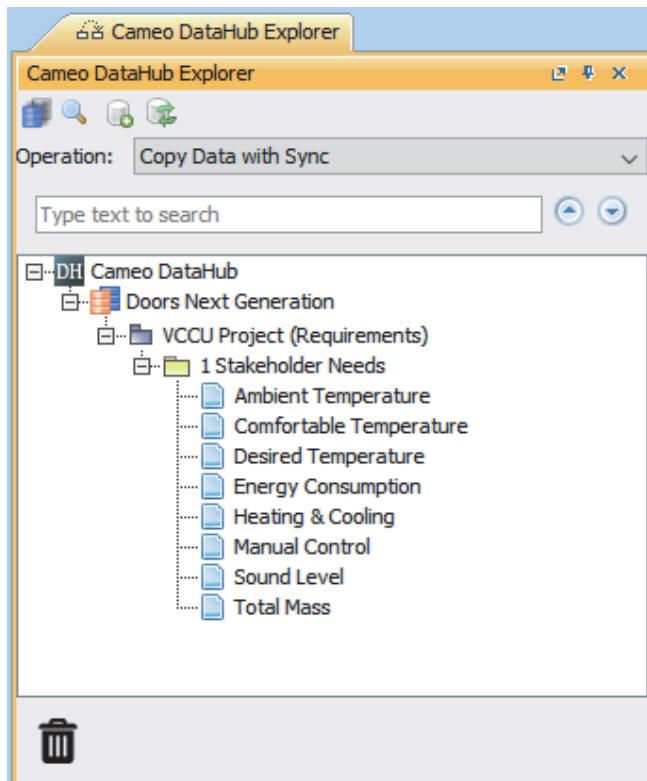
1. Be sure you have Cameo DataHub Plugin 19.0 or later installed on top of the [modeling tool](#).
2. Connect to the DOORS NG 6.x repository:
 - a. From the main menu of the [modeling tool](#), select **Tool > DataHub > DataHub Explorer**. The **Cameo DataHub Explorer** panel opens to the right of the [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 *VCCU Project (Requirements)*.



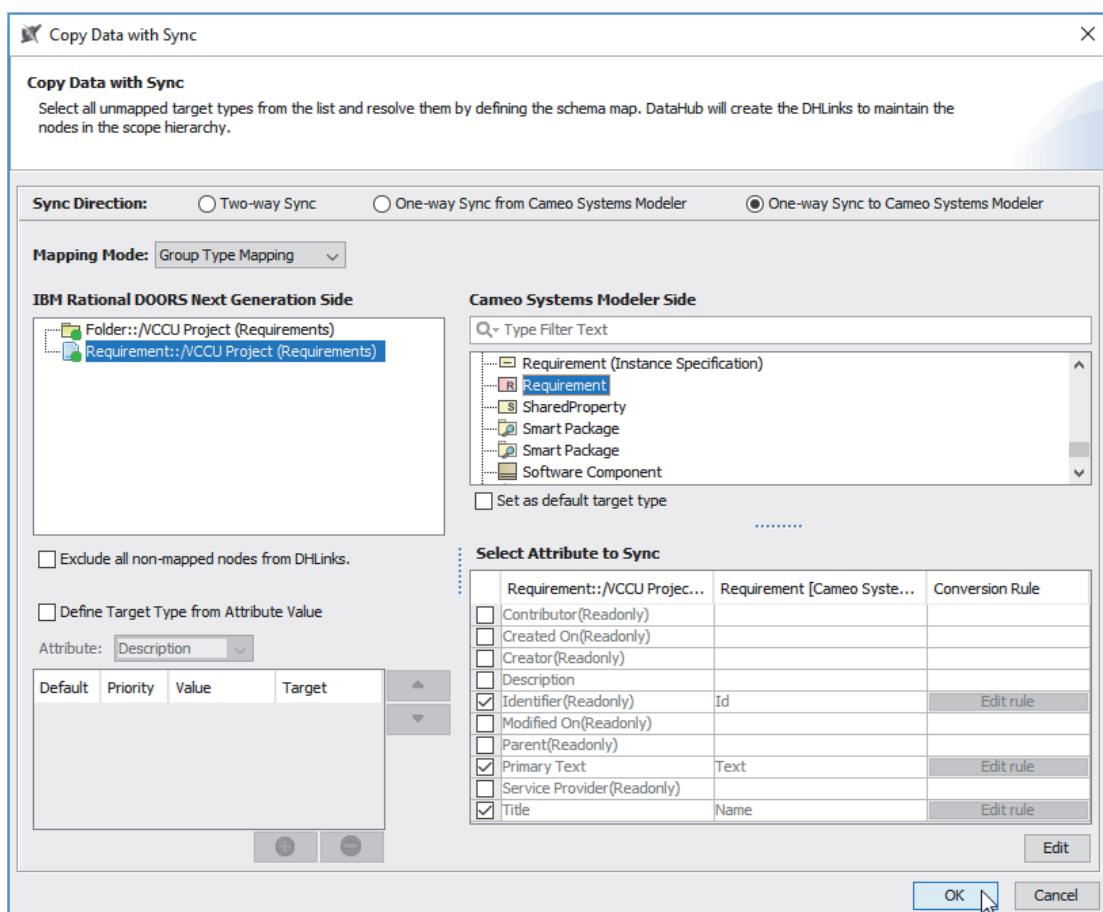
3. Synchronize data:

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

i In this case, you should ignore the inner package, *1 Stakeholder Needs*, created in [step 1 of this cell 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 risk missing the items that will be created in the DOORS project afterwards. In that case, only individual items are synchronized, not the whole project.

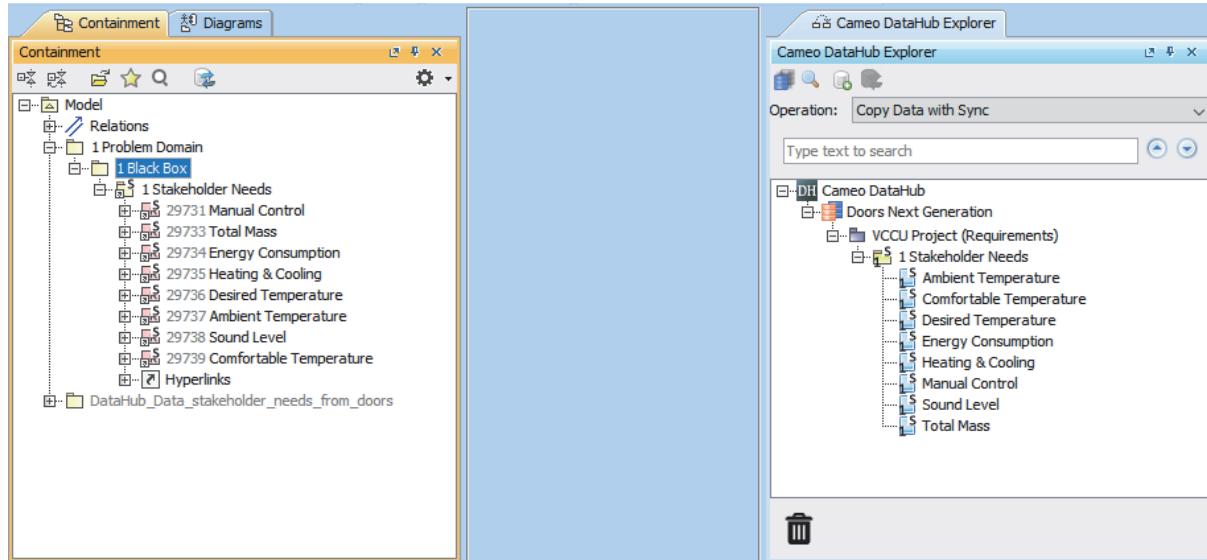
b. In the open dialog, do the following:

- i. Click **One-way Synch to Cameo Systems Modeler** to select the synchronization direction.
- ii. Specify the mappings:
 1. Folder to Package
 2. Requirement to Requirement
- iii. Click the **OK** button.



- c. Wait for the message announcing that synchronization is completed. Synchronized items in both trees become marked with *S* (see the following figure).
4. Expand the contents of the *1 Black Box* package and see the *Stakeholder Needs* package with all its data imported into your model.

i "S" indicates synchronized data, and "1" the synchronization source. As you have selected **One-way Synch to Cameo Systems Modeler**, the DOORS NG 6.x becomes the single source for synchronization.



Step 4. Grouping stakeholder needs

i This does not apply if stakeholder needs are synchronized from external software for requirements management (for example, IBM® Rational® DOORS®), because all the changes you make in the SysML model will be overwritten after routine synchronization. Grouping, numbering, and categorization of stakeholder needs in that case should be performed in the requirements management tool.

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, industry standards, or business requirements, among others.

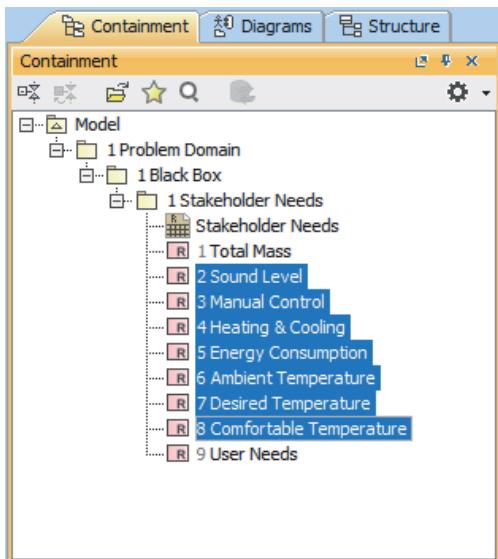
For grouping stakeholder needs, you need to create extra elements of the SysML requirement type, each representing a single group of stakeholder needs and generally referred to as a grouping requirement. A grouping requirement might not have any text itself, only a unique identification number and name.

Now let's analyze the stakeholder needs of the Vehicle Climate Control Unit in your model. Actually, the majority of them are user needs, and therefore can be grouped under the *User Needs* requirement. The stakeholder need captured as the *Total Mass* requirement sounds like a design constraint. For this reason, it should appear under the *Design Constraints* requirement. The top-level grouping requirement simply named *Stakeholder Needs* can also be created, to complete the requirements hierarchy.

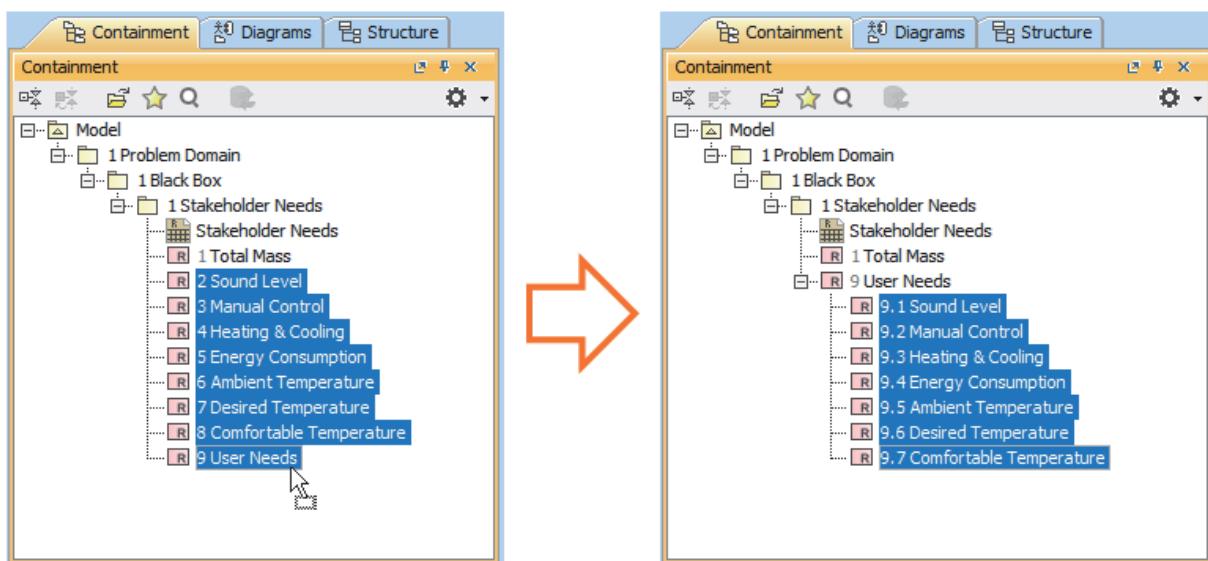
To group stakeholder needs

1. Create the *User Needs* 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 the items except the *Total Mass* requirement (the first item in the list) and the *User Needs* requirement (the last item in the list).

- i** 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.

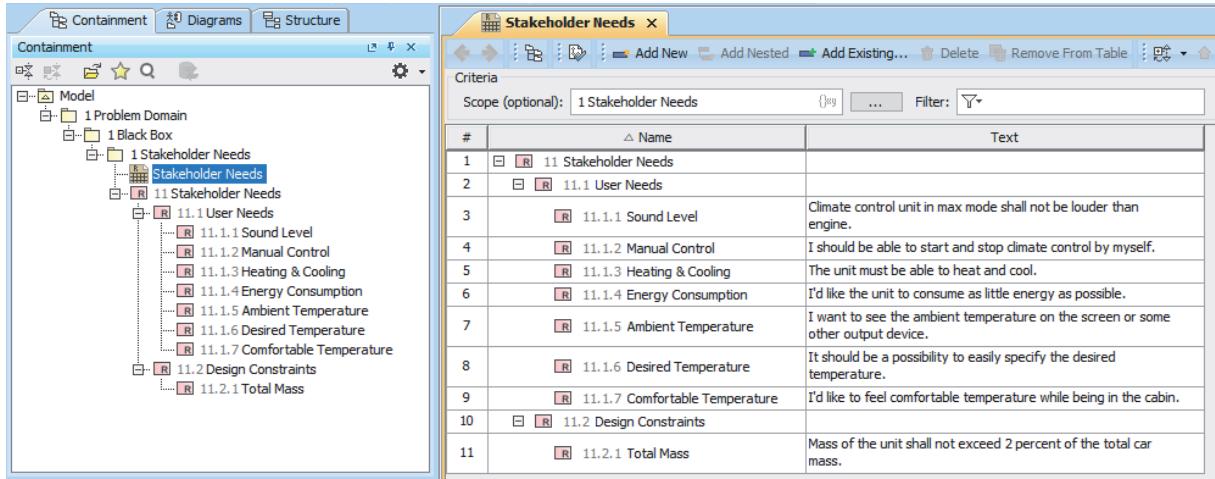


3. Drag the selection onto the *User Needs* requirement (see the figure on the left). It becomes a grouping requirement (see the figure on the right).



4. Repeat the step 1 to create the *Design Constraints* requirement.
5. Select the *Total Mass* requirement and drag it onto the *Design Constraints* requirement. The latter becomes a grouping requirement.
6. Repeat step 1 to create the *Stakeholder Needs* requirement.
7. Select both grouping requirements: *User Needs* and *Design Constraints*.
8. Drag the selection onto the *Stakeholder Needs* requirement. It becomes the top-level grouping requirement. See appropriate changes in the SysML requirement table.

- i** 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

i This does not apply if stakeholder needs are synchronized from external software for requirements management (for example, IBM® Rational® DOORS®), because all the changes you make in the SysML model will be overwritten after the routine synchronization. Grouping, numbering, and categorization of stakeholder needs in that case should be performed in the requirements management tool.

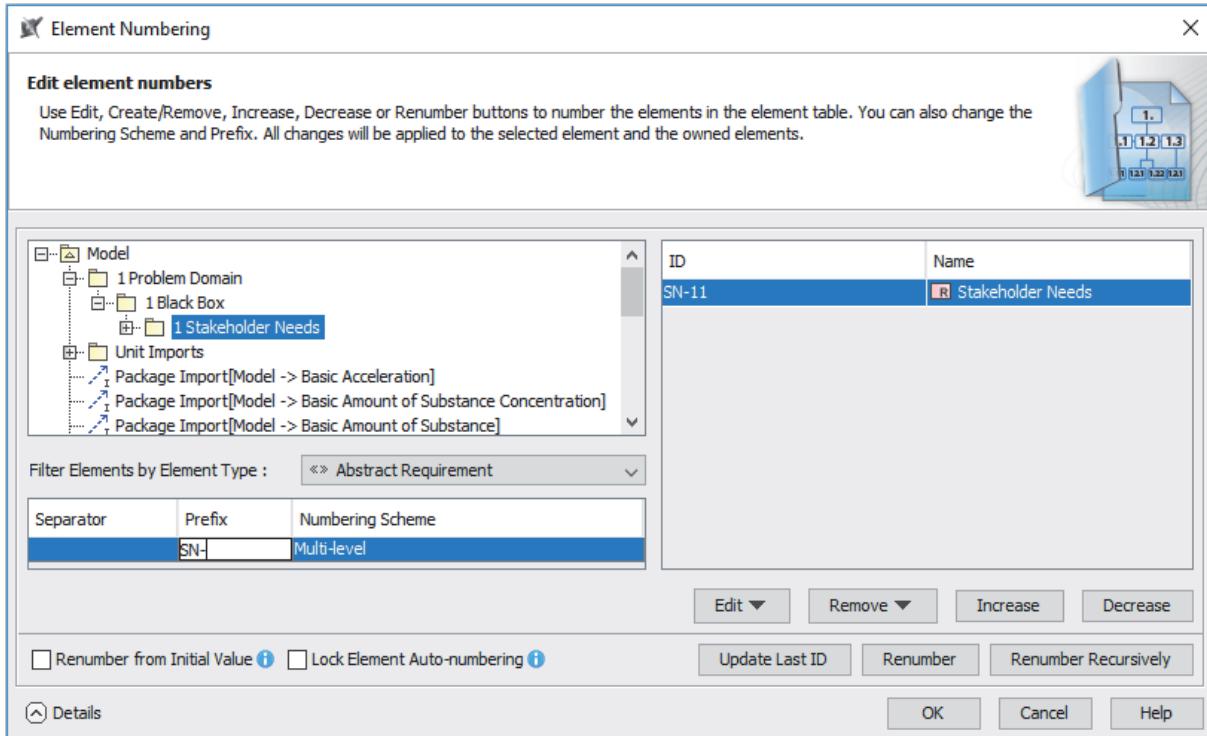
SysML requirements, which we utilize for capturing and storing stakeholder needs, may have hierarchical numbers. The hierarchical number should not be mixed with the unique identifier, which can never be changed.

Stakeholder needs are by default numbered using natural numbers, from 1 to infinity. In order to distinguish stakeholder needs from system or implementation requirements, we recommend numbering them with special prefixes (i.e., *SN-*, where *S* stands for *stakeholder* and *N* for *needs*).

After organizing stakeholder needs into groups, their numbering became inaccurate, as it now begins from 11. Let's fix this, too.

To number stakeholder needs with the prefix *SN-*

1. In the Model Browser, right-click *Stakeholder Needs* (the top-level grouping requirement), 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-* (see the following figure), and press Enter. The hierarchical number of the *Stakeholder Needs* requirement is updated to *SN-11*.



3. Click **Details** to see more options, if they are not seen yet.
4. Click to select the **Renumber From Initial Value** check box.
5. Click the **Renumber Recursively** button. Every item of stakeholder needs becomes numbered with the prefix *SN-* starting from number 1.

(i) If you want to make sure that all the items are renumbered, expand the model tree on the left side of the dialog to see the entire hierarchy of the stakeholder needs for the VCCU.

6. Click **OK** to close the dialog.

#	△ Name	Text
1	□ R SN-1 Stakeholder Needs	
2	□ R SN-1.1 User Needs	
3	R SN-1.1.1 Sound Level	Climate control unit in max mode shall not be louder than engine.
4	R SN-1.1.2 Manual Control	I should be able to start and stop climate control by myself.
5	R SN-1.1.3 Heating & Cooling	The unit must be able to heat and cool.
6	R SN-1.1.4 Energy Consumption	I'd like the unit to consume as little energy as possible.
7	R SN-1.1.5 Ambient Temperature	I want to see the ambient temperature on the screen or some other output device.
8	R SN-1.1.6 Desired Temperature	It should be a possibility to easily specify the desired temperature.
9	R SN-1.1.7 Comfortable Temperature	I'd like to feel comfortable temperature while being in the cabin
10	□ R SN-1.2 Design Constraints	
11	R SN-1.2.1 Total Mass	Mass of the unit shall not exceed 2 percent of the total car mass.

Step 6. Categorizing stakeholder needs

i This does not apply if stakeholder needs are synchronized from external software for requirements management (for example, IBM® Rational® DOORS®); all the changes you make in the SysML model will be overwritten after the routine synchronization. Grouping, numbering, and categorization of stakeholder needs should be performed in the requirements management tool.

Stakeholder needs can be either functional or non-functional. Functional stakeholder needs specify what the **Sol** is expected to do. They are further refined by use cases and use case scenarios (see Chapter [Use Cases](#)). Non-functional stakeholder needs that identify quantifiable characteristics of the **Sol** are refined by measures of effectiveness (see [Measures of Effectiveness](#)).

Categorization facilitates further analysis of stakeholder needs, especially when dealing with a large scope of information. This step may be skipped if the case does not require it.

While non-functional stakeholder needs can be captured as SysML requirements, functional stakeholder needs should be converted to SysML functional requirements. In technical terms, functional requirement (together with interface, performance, design, and other more specific requirements, which are not needed thus far) is a sub-type of the requirement and inherits all its features.

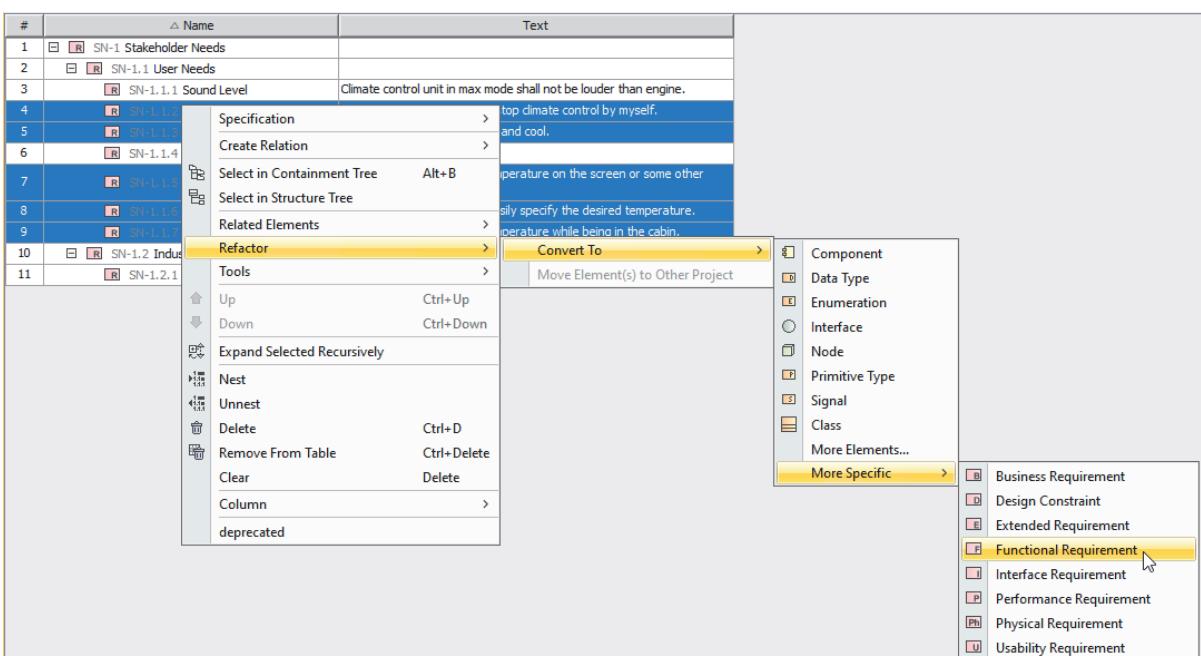
In the given sample, the majority of stakeholder needs (except *Sound Level*, *Energy Consumption*, and *Total Mass*) are functional. Therefore, they should be refactored to SysML functional requirements.

To convert SysML requirements to functional requirements

1. In the Model Browser or on the requirements table, select all requirements except *Sound Level*, *Energy Consumption*, and *Total Weight*.

i To select a set of non-adjacent items in the tree, click them one by one while holding down the Ctrl key.

2. Right-click the selection and then choose **Refactor > Convert To > More Specific > Functional Requirement**.



As a result, all the requirements are converted to functional, and *F* instead of *R* is displayed on each element icon.

#	△ Name	Text
1	⊖ R SN-1 Stakeholder Needs	
2	⊖ R SN-1.1 User Needs	
3	R SN-1.1.1 Sound Level	Climate control unit in max mode shall not be louder than engine.
4	F SN-1.1.2 Manual Control	I should be able to start and stop climate control by myself.
5	F SN-1.1.3 Heating & Cooling	The unit must be able to heat and cool.
6	R SN-1.1.4 Energy Consumption	I'd like the unit to consume as little energy as possible.
7	F SN-1.1.5 Ambient Temperature	I want to see the ambient temperature on the screen or some other output device.
8	F SN-1.1.6 Desired Temperature	It should be a possibility to easily specify the desired temperature.
9	F SN-1.1.7 Comfortable Temperature	I'd like to feel comfortable temperature while being in the cabin
10	⊖ R SN-1.2 Design Constraints	
11	R SN-1.2.1 Total Mass	Mass of the unit shall not exceed 2 percent of the total car mass.

Stakeholder Needs done. What's next?

- Stakeholder needs are analyzed to identify system contexts (see Chapter [System Context](#)).
- Functional stakeholder needs are further analyzed and refined by use cases and use case scenarios (see Chapter [Use Cases](#)).
- Non-functional quantifiable stakeholder needs are refined by measures of effectiveness (see Chapter [Measures of Effectiveness](#)).

System Context

What is it?

System context (or operating environment) determines an external view of the system. The system context introduces all external entities that do not belong to the system but do interact with it. More than one system context can be defined for a single **Sol**. Besides the system itself (currently considered to be a black box), the collection of elements in the particular system context can include external systems (natural or artificial) and users (humans, organizations) that interact with the **Sol** to exchange data, matter, energy, or even human resources.

In overall, this cell produces:

- Definitions of system contexts
- Participants of each system context: **Sol**, supposed users of the system, other systems, etc.
- Interactions between participants of each system context
- Items that are exchanged over these interactions

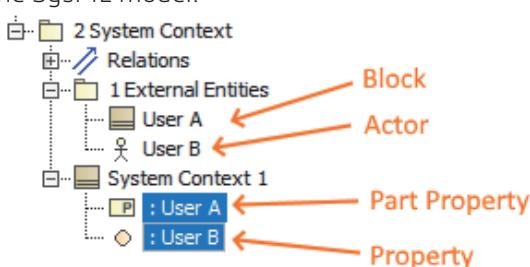
Who is responsible?

System contexts can be identified and specified by the Systems Analyst or Systems Engineer.

How to model?

System context diagram pictures the **Sol** at the center, with no details of its interior structure, surrounded by all its interacting entities, such as supposed users and external systems to name a few. In the SysML model, system contexts can be captured as elements of the SysML block type. All the entities can be captured as SysML blocks, too. As defined by SysML, once the entity block is used in the system context block, a part property typed by the former, is created for the latter.

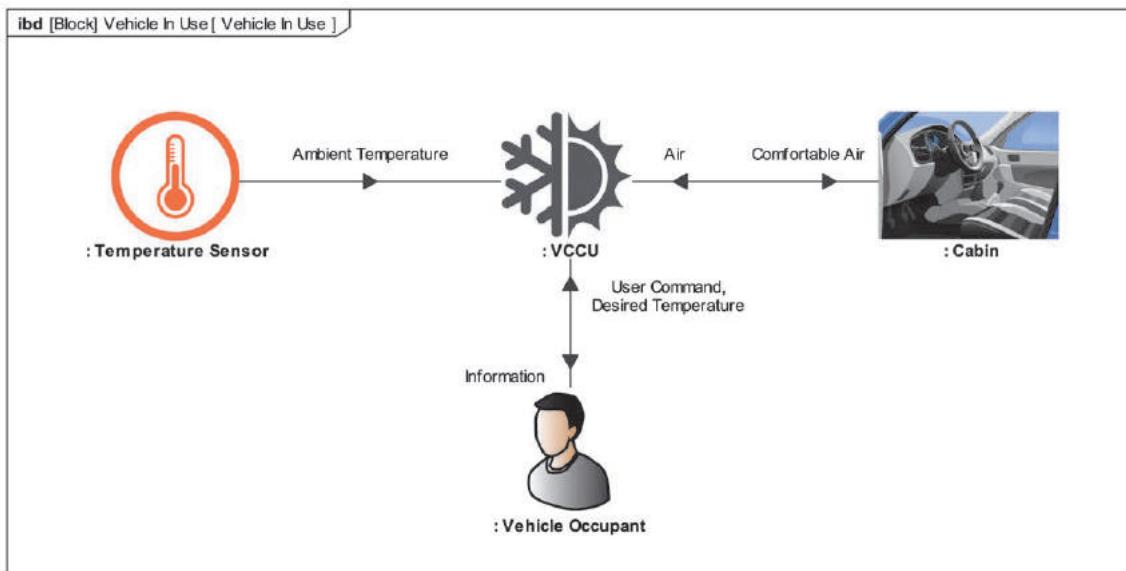
i We recommend using blocks rather than actors (stick man notation) to capture even human-nature entities (for example, users of the **Sol**) because the actor cannot be a part of the system context in the SysML model.



A **SysML internal block diagram (ibd)** can serve to represent the entities of the particular system context, and interactions between them. Each entity is represented as a part property of the related system context block.

Interactions within the system context can be captured as connectors with one or more item flows between those part properties. Data, matter, or energy are items that can flow. They can be stored in the model as signals, blocks, or value types, depending on their nature, and in general are referred as exchange items.

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

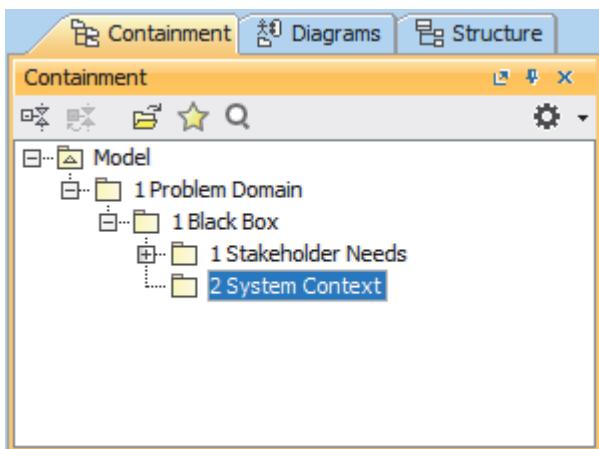


Tutorial

- Step 1. Organizing the model for the system context
- Step 2. Capturing system contexts
- Step 3. Creating an ibd for a system context
- Step 4. Capturing participants of the system context
- Step 5. Specifying interactions between the participants of system context
- Step 6. Adding item flows to the system context

Step 1. Organizing the model for the system context

Following the structure of the MagicGrid framework, system contexts and owned-by-them SysML internal block diagrams, together with the 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 System Context*.



To organize the model for the system context

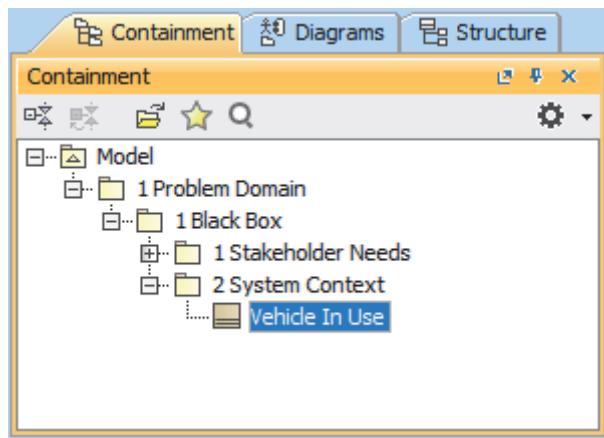
1. Right-click the *1 Black Box* package and select **Create Element**.
2. In the search box, type *pa* (the first two letters of the element type *Package*) and press Enter.
3. Type *2 System Context* to specify the name of the new package and press Enter.

Step 2. Capturing system contexts

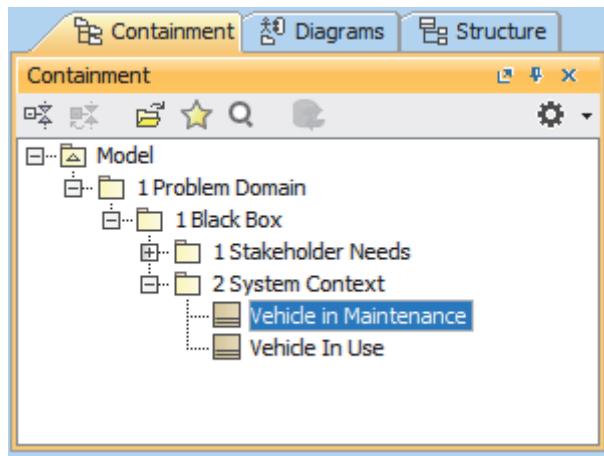
System contexts can be identified by analyzing stakeholder needs (see Chapter [Stakeholder Needs](#)). Stakeholder needs of the Vehicle Climate Control Unit enable you to identify a single context: the *Vehicle In Use*. It can be captured in your model as an element of the SysML block type.

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

1. Right-click the *2 System Context* package and select **Create Element**.
2. In the search box, type *bl* (the first 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.



It is important to understand that the **Sol** may operate in more than one system context. To illustrate this, we need to capture one more system context. In this example, we will use the *Vehicle in Maintenance* (see the following figure). You can create the appropriate block on your own by following the procedure above.



The recently captured system context is not included in further analysis. It is not mentioned in the stakeholder needs and has been created for demonstration purposes only.

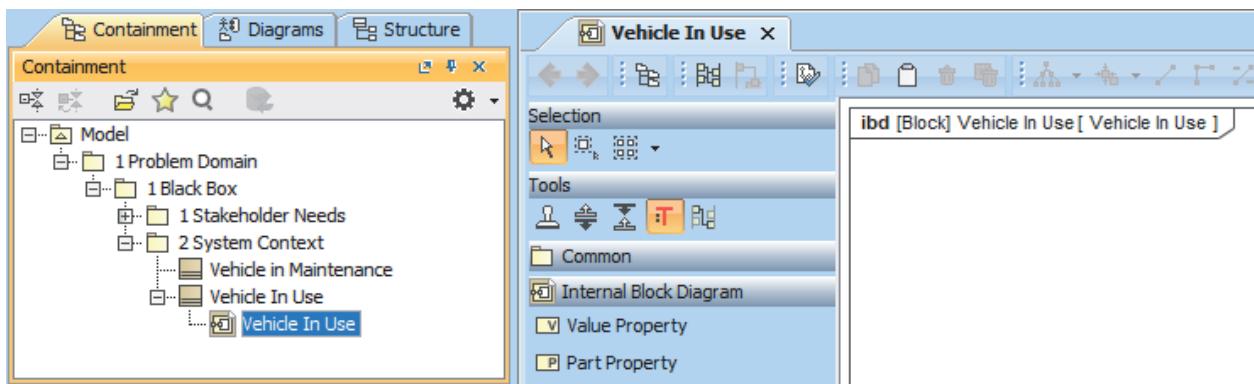
Step 3. Creating an ibd for a system context

To specify 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 double-press Enter. The diagram is created and represented in the Model Browser under the *Vehicle In Use* block.

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



Step 4. Capturing participants of the system context

Entities that participate in the system context can be captured in the model as the part properties of that system context, where each part property is typed by the block, that captures the appropriate entity (for example, the **Sol**, its supposed users, or some external system).

i 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 Sol* defined as a block can type a part property of the *Vehicle in Use* system context and part property of the *Vehicle in Maintenance* system context, meaning that it participates in both.

Stakeholder needs indicate that the *Vehicle In Use* system context includes the following participants:

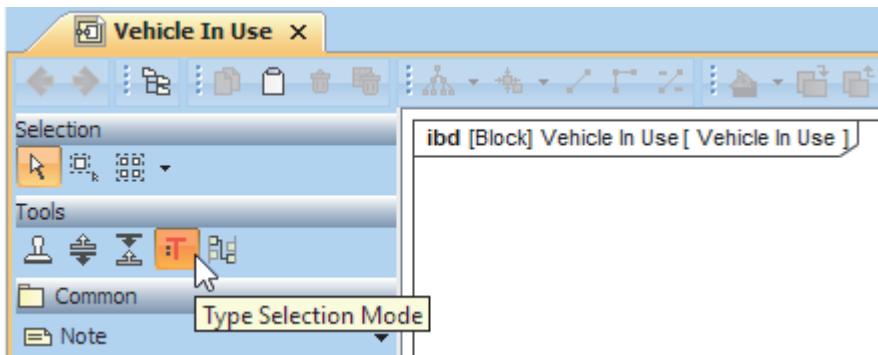
- Vehicle Climate Control Unit, the **Sol**
- Vehicle Occupant (driver or passenger), the supposed user of the **Sol**
- Vehicle Cabin, the closed environment that must be provided with a comfortable air temperature
- Temperature Sensor, the external system to provide an ambient temperature to the **Sol**

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

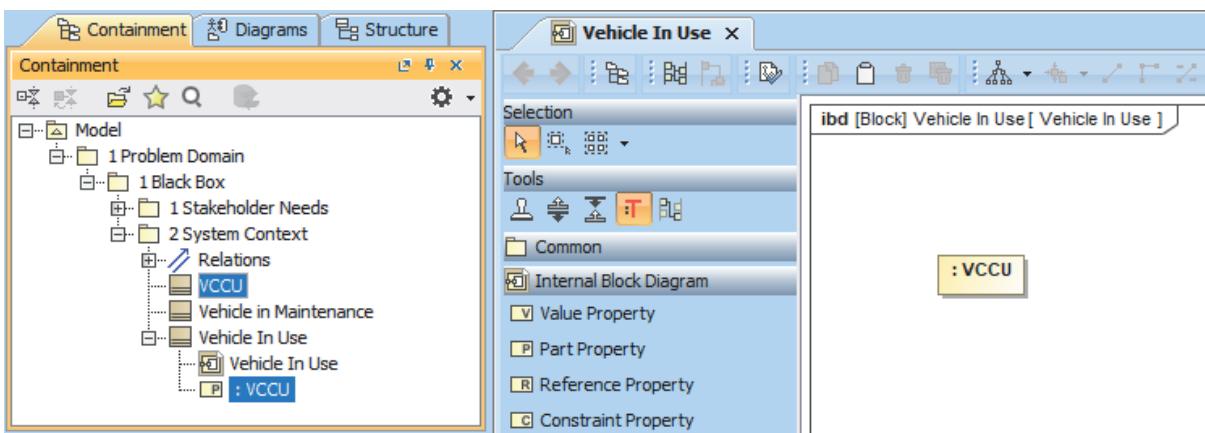
1. Open the ibd created in [step 3 of this cell tutorial](#), if not yet opened.

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

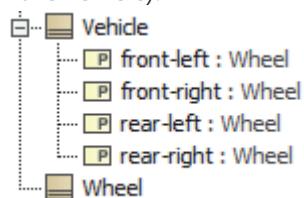
i In this mode, when you create a part property, the tool offers you a list of blocks, from which you can choose the part property type. If the necessary block does not exist yet, you can simply create it by typing its name directly on the part property shape.



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 *VCCU* next to the colon (":") directly on the part property shape and press Enter. The *VCCU* block to type the just-created part property emerges in the Model Browser.



i When the Type Selection Mode is enabled, typing on the part property shape always defines the name of the new block. This new block 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, names are not necessary, since part properties created in this diagram can be easily identified by their types. Names are useful when you want to specify more than one usage of the same block (for example, the front-left, front-right, rear-left, and rear-right wheels of the vehicle).

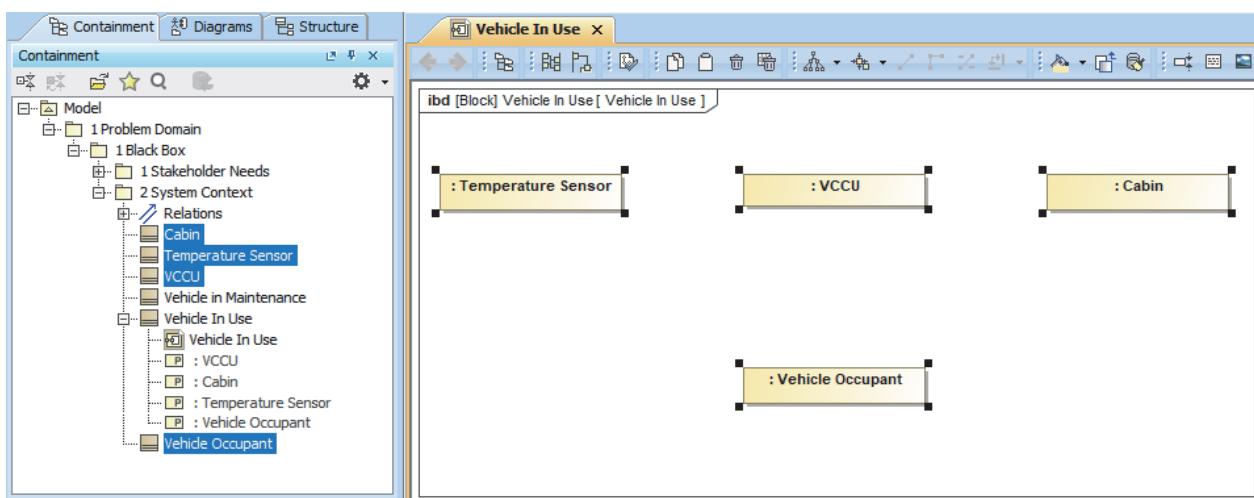


5. Repeat steps 3 and 4 to create the rest of part properties: one typed by the *Vehicle Occupant* block, one by the *Cabin* block, and one by the *Temperature Sensor* block.

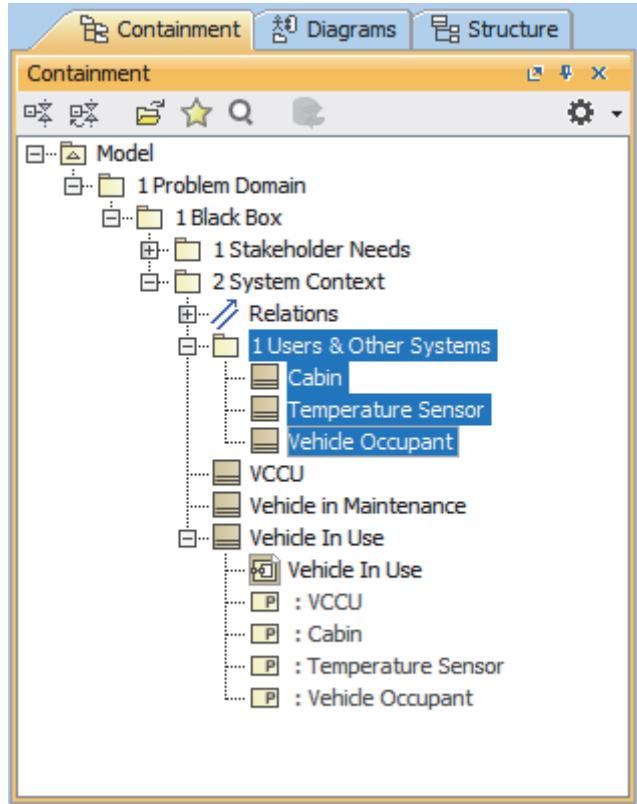
i Since you need to create more than one part property in succession, use the functionality of the Sticky button (see the following figure), so it is not necessary to click the **Part Property** button on the palette each time you want to create a new part property. When the Sticky mode is enabled, you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



When you are finished, you have four part properties and the same number of blocks that type them in your model.



To keep your model well-organized, we recommend storing the blocks that capture users and other systems in a separate package within the *2 System Context* package. For this, you need to create an internal package and drag the blocks to the new location in the Model Browser. Hold down the Shift key to select more than one block.



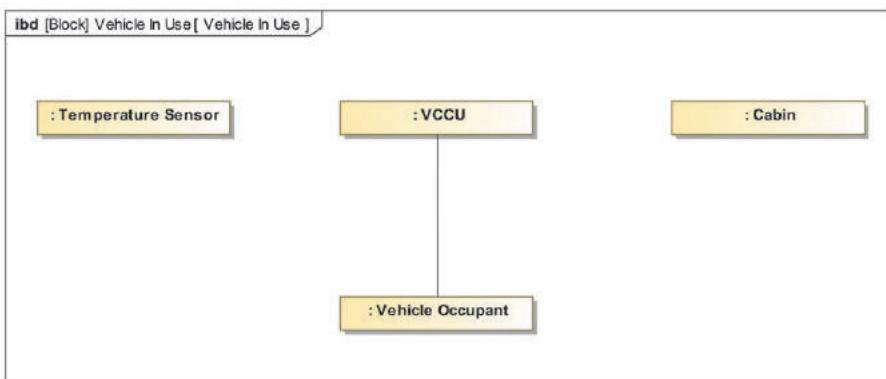
Step 5. Specifying interactions between the participants of system context

The stakeholder needs indicate that the Vehicle Climate Control Unit interacts with all participants of the *Vehicle In Use* system context. We need to capture this information in the model.

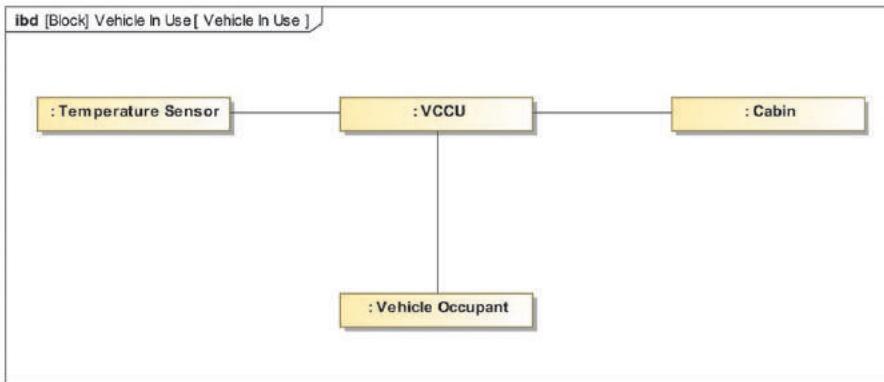
Remember that interactions can be specified as connectors with one or more item flows between appropriate part properties. We will start with drawing the connectors.

To draw a connector between the *:VCCU* and *:Vehicle Occupant* part properties

1. Select the *:VCCU* part property and click the Connector button on its smart manipulator toolbar.
2. Select the *:Vehicle Occupant* part property. The connector is created.

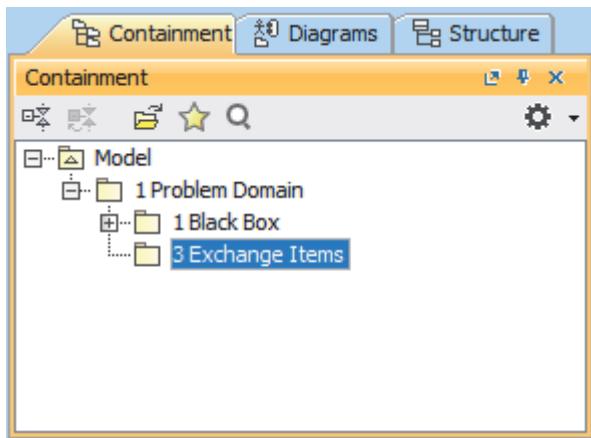


You can create other connectors using this same method. Once you have finished, your ibd should look very similar to the one below.



Step 6. Adding item flows to the system context

To begin, we first need the items that can flow from one part property to another across the connector and back. We recommend storing the elements that capture these items in a separate package (see the following figure). The contents of this package will be used in the white-box analysis model, too. Therefore, instead of creating the new package directly under the *2 System Context* package (which is a part of the black-box analysis model) create it under the *1 Problem Domain* package.



If we assume that you have created the *3 Exchange Items* package (the package number is 3 because number 2 will be used for the *White Box* package) on your own, we can move directly to creating items that can flow and assigning item flows to the connectors. One or more item flows can be assigned on a single connector.

Stakeholder needs indicate that:

- Vehicle Occupant shall be able to turn the VCCU on and off, and specify the desired temperature (see *SN-1.1.2, SN-1.1.6*)
- VCCU shall provide a comfortable air temperature for the Vehicle Occupant sitting in the Cabin (see *SN-1.1.7*)
- VCCU shall obtain the ambient temperature from the Temperature Sensor of the vehicle (see *SN-1.1.5*)
- VCCU shall display status and temperature information (see *SN-1.1.5*)

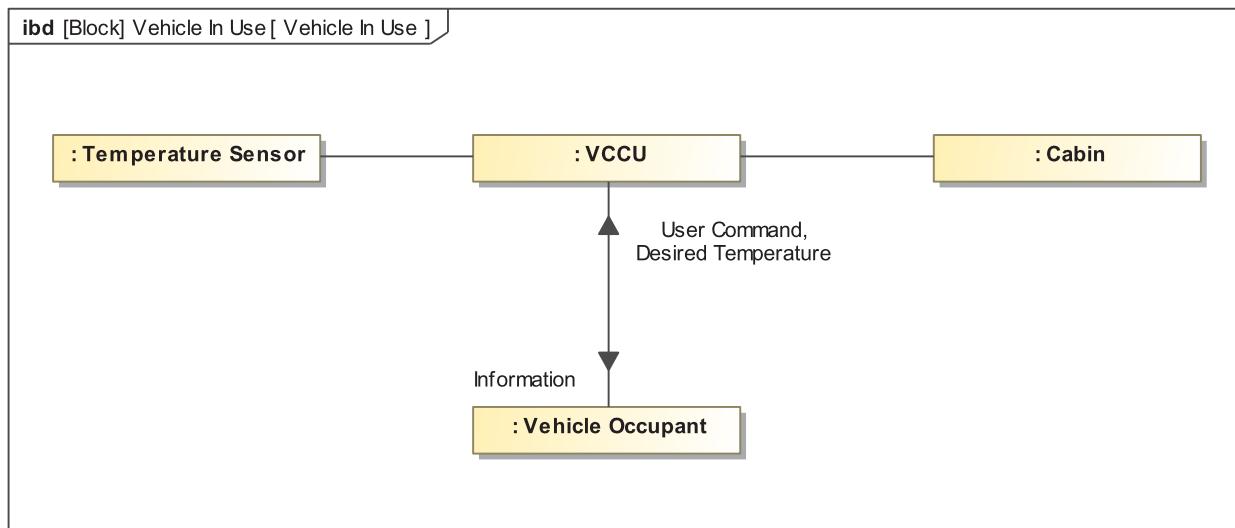
These statements enable you to identify the following exchange items:

- User Command
- Desired Temperature
- Air
- Comfortable Air
- Ambient Temperature
- Information

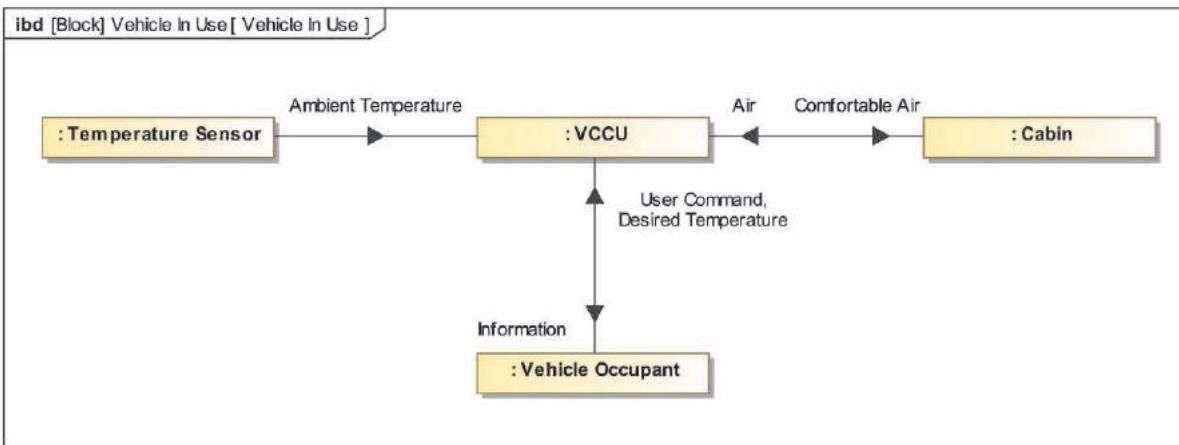
Air and Comfortable Air are matter; thus, they can be captured in the model as blocks. The rest of the items can be captured as signals, because they mean data.

To capture the User Command, Desired Temperature, and Information in the model and specify them as items conveyed by the item flows between the `:VCCU` and `:Vehicle Occupant` part properties

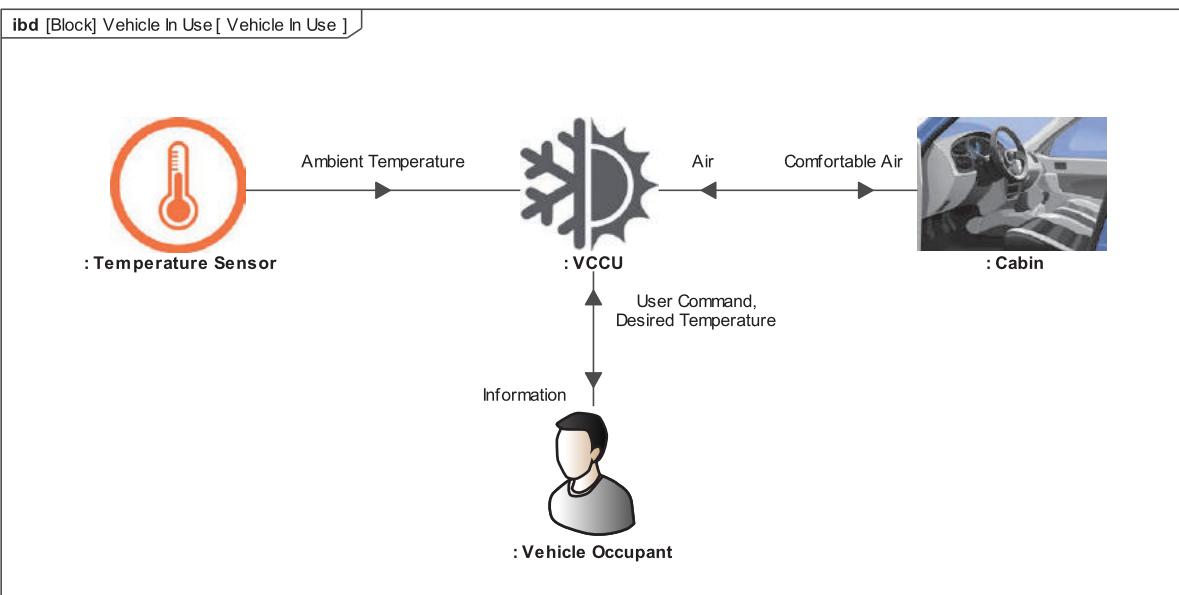
1. Right-click the `3 Exchange Items` package and select **Create Element**.
2. In the search box, type `si` (the first two letters of the element type *Signal*) and press Enter.
3. Type *User Command* to specify the name of the new signal and press Enter.
4. Drag the signal to the connector between `:VCCU` and `:Vehicle Occupant` part properties in the *Vehicle In Use* ibd.
5. In the open dialog, select **From Vehicle Occupant To VCCU** as the direction of the flow and click **Finish**. A new item flow conveying the *User Command* signal is created and displayed on the connector.
6. Repeat steps 1 to 3 to create the *Desired Temperature* signal.
7. Drag the signal to the same connector.
8. In the open dialog, select **From Vehicle Occupant To VCCU** as the direction of the flow and click **Finish**. A new item flow conveying the *Desired Temperature* signal is created and displayed on the connector.
9. Repeat steps 1 to 3 to create the *Information* signal.
10. Drag the signal to the same connector.
11. In the open dialog, click **Finish**. A new item flow conveying the *Information* signal is created and displayed on the connector.



In the same manner, you can capture the remaining exchange items and then specify item flows on other connectors in the *Vehicle In Use* ibd. Once finished, your ibd should look very similar to the one below.



As you can see in the figure below, system context diagrams can be supplemented with various images so 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](#).



System Context done. What's next?

- System contexts are used to analyze the expected behavior of the **Sol** (see Chapter [Use Cases](#)). Behavior analysis begins with identifying the use cases of the **Sol**, and each use case belongs to one or more system contexts of that **Sol**.
- Participants of each system context captured as part properties of that system context are also used in the [Use Cases](#) cell. They can be represented as swimlanes in the SysML activity diagram that describes a use case scenario.
- Exchange items identified in the System Context cell are used in the use case scenario as object flow items and pin types.
- 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 [Measures of Effectiveness](#).

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. Each use case must belong to one or more system contexts defined in the System Context cell (see Chapter [System Context](#)).

Overall, this cell produces:

- Functional use cases that provide meaningful value to the user.
- Use case scenarios on how the system is expected to interact with the user and/or other systems.

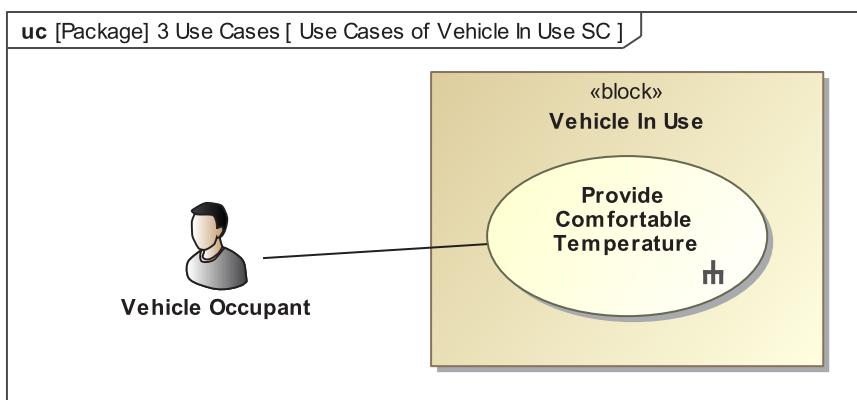
Who is responsible?

Use cases and their scenarios can be specified by the Systems Analyst or Systems Engineer.

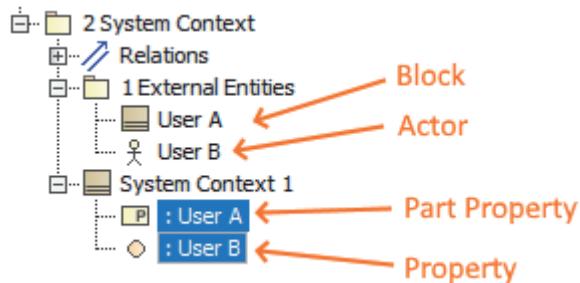
How to model?

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 the System Context cell (see Chapter [System Context](#)). Use cases can be captured as elements of the use case type. A single use case can be performed in different contexts.

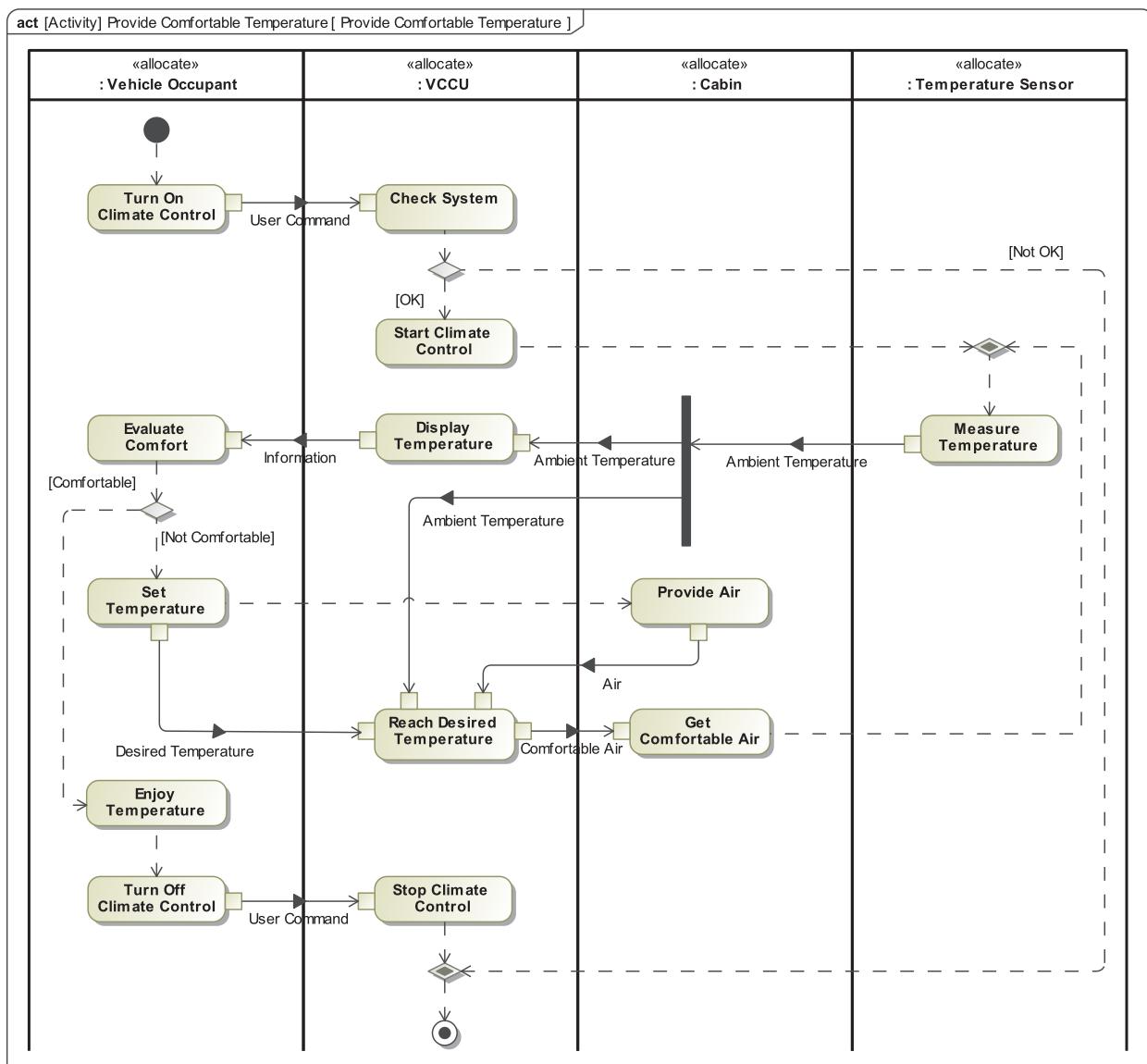
Entities that perform one or more use cases of the system context should be represented as blocks in the use case diagram. Here you should remember that these blocks type part properties of the system context. As you can see in the following figure, even humans, such as the vehicle occupant, can be represented as blocks (with or without images applied).



- i** We recommend using blocks rather than actors (stick man notation) to capture even human-nature entities (for example, users of the **Sol**) because the actor cannot be a part of the system context in the SysML model.



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, **Sol**, supposed users of the system, and/or other systems can be represented as swimlanes (activity partitions). In the sequence diagram, they are represented as lifelines. The **Sol** is meanwhile considered a black box.

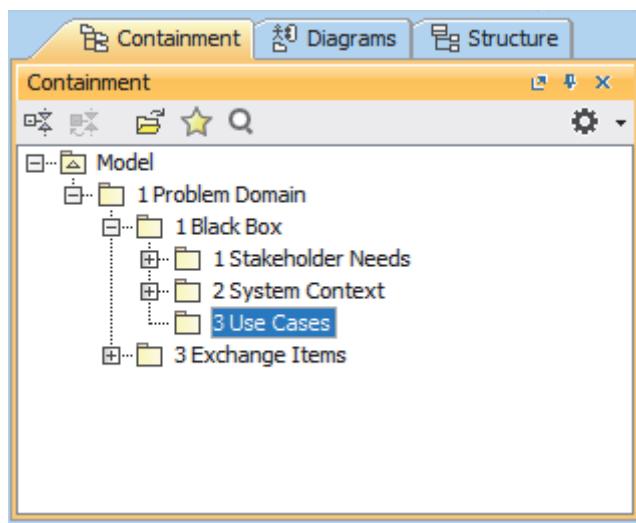


Tutorial

- Step 1. Organizing the model for use cases
- Step 2. Creating a diagram for use cases
- Step 3. Capturing use cases
- Step 4. Creating a diagram for specifying the use case scenario
- Step 5. Creating swimlanes and setting allocation mode
- Step 6. Specifying steps of the use case scenario
- Step 7. Adding item flows to the use case scenario
- Step 8. Supplementing the use case scenario with a parallel flow

Step 1. Organizing the model for use cases

Following the structure of the MagicGrid framework, use cases and their scenarios in the form of SysML activity or sequence diagrams, 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 Use Cases*.



To organize the model for the use cases

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 Use Cases* to specify the name of the new package and press Enter.

Step 2. Creating a diagram for use cases

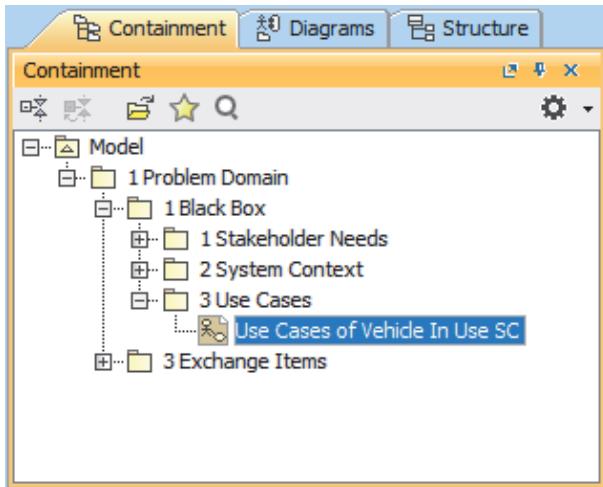
You must create a SysML use case diagram to begin capturing use cases of the *Vehicle In Use* system context (see Chapter [System Context](#)). You need to display the shape of the block that captures the system context on the diagram pane as well. The *Vehicle Occupant* block, which represents the user of the *Sol*, 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 *3 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.

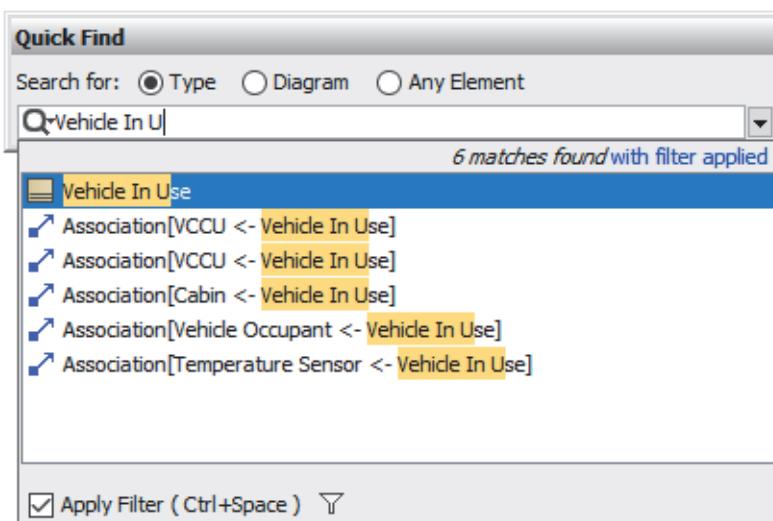
(i) If you have more than one system context, we recommend storing use cases of each system context in a separate package. In this case, you do not need to.

3. Type *Use Cases of Vehicle In Use SC* to specify the name of the new diagram and press Enter again.

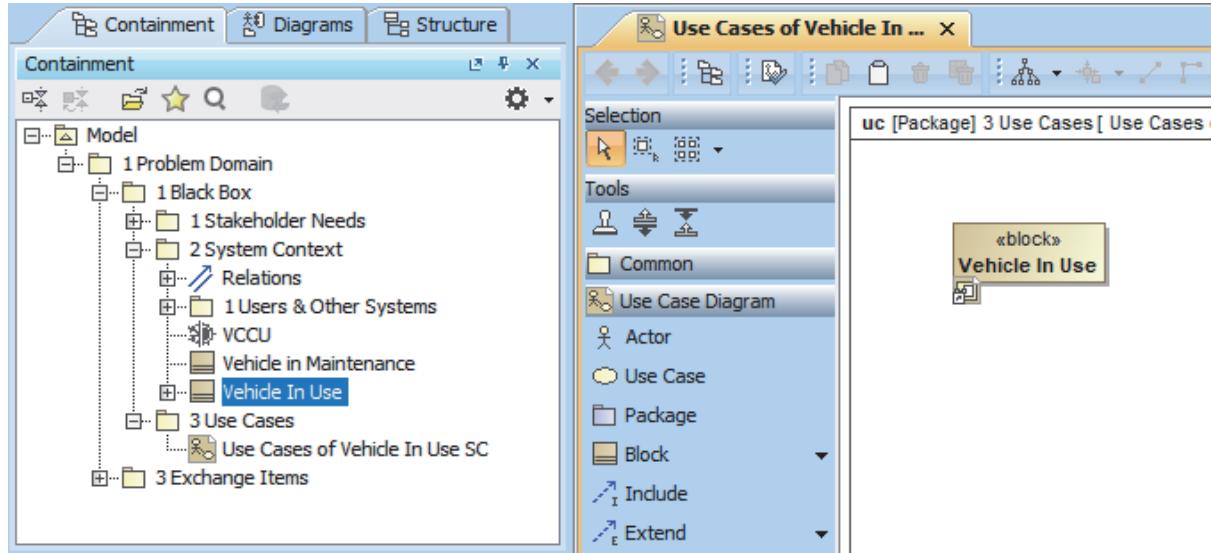


4. In the Model Browser, select the *Vehicle In Use* block. For this, use the quick find capability:

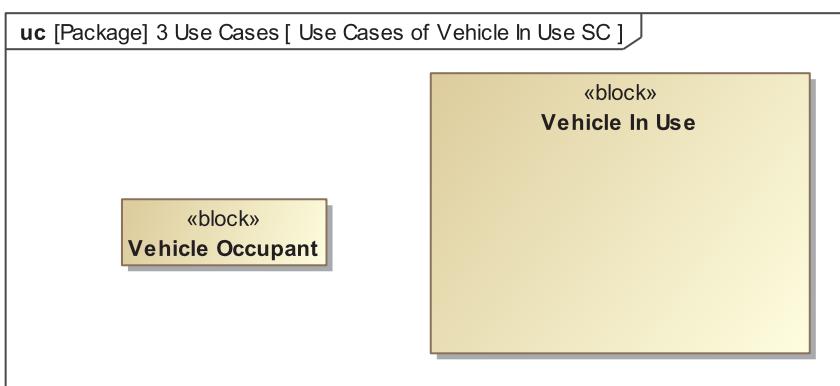
- a. Press **Ctrl + Alt + F**. The **Quick Find** dialog opens.
- b. Type *Vehicle In U*.
- c. When you see the *Vehicle In Use* block selected in the search results list, press Enter, as shown in the following figure. The *Vehicle In Use* block is selected in the Model Browser.



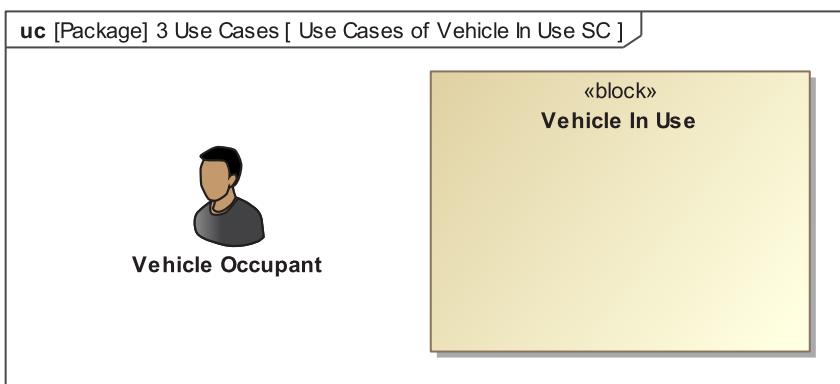
5. Drag the *Vehicle In Use* block to the newly created use case diagram pane. The shape of the *Vehicle In Use* block appears on the diagram.



6. Repeat the sub-steps of Step 4 to find 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.
7. If the relationship is displayed on the diagram too, select it and press Delete to remove its symbol from the diagram. When the dialog asks if you want to remove the relationship from the model, click **No**.
8. Select the *Vehicle In Use* block and drag the bottom right corner of the symbol to enlarge it. It must be larger to nest the use cases inside.



As you can see in the figure below, the *Vehicle Occupant* block can be displayed as an image you applied on that block in the ibd of the *Vehicle In Use* system context (see Chapter [System Context](#)). For this, select the block and click the Show As Image button on the diagram toolbar.

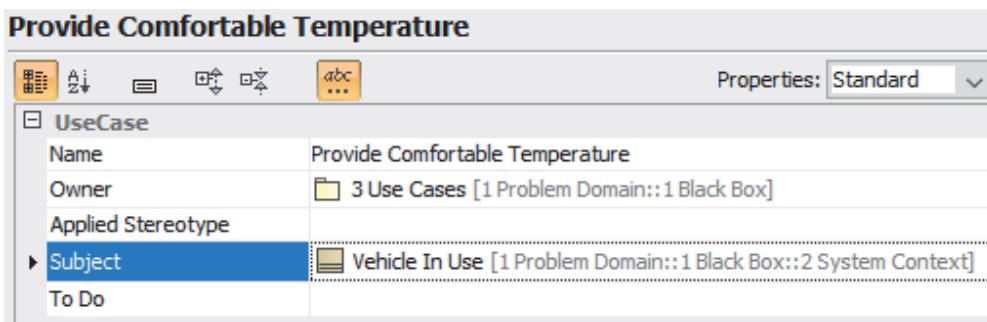


Step 3. Capturing use cases

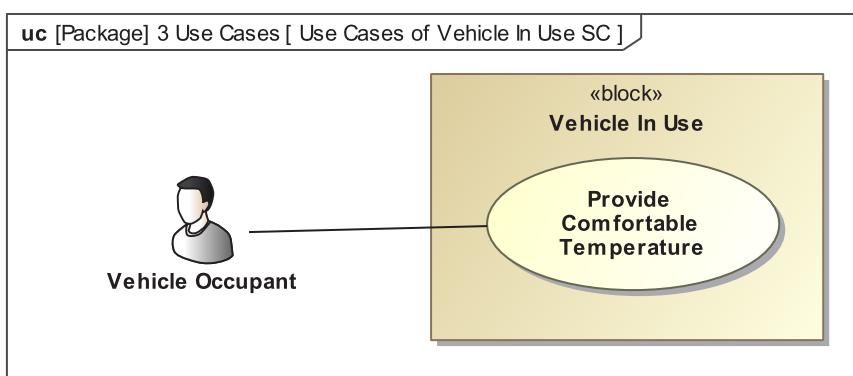
By analyzing stakeholder needs, we can see that the main goal of the VCCU is to provide a comfortable temperature for all vehicle occupants while they are in the cabin. This goal can be captured in the model as the *Provide Comfortable Temperature* use case of the *Vehicle In Use* system context. Let's capture it in the *Use Cases of Vehicle In Use SC* diagram.

To capture the *Provide Comfortable Temperature* use case

1. Open the use case diagram created in [step 2 of this cell 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 *Provide 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 *Vehicle Occupant* block (with or without image applied) that represents the user of the VCCU, click the Association button on its smart manipulator toolbar, and then select the *Provide Comfortable Temperature* use case to relate these two elements.



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

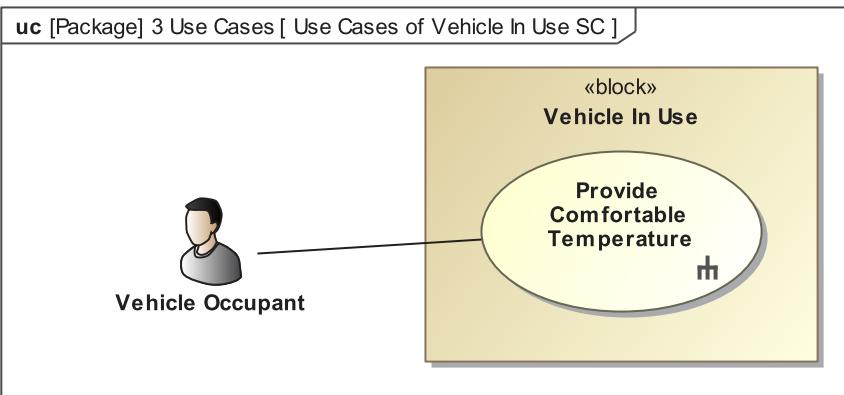
A use case scenario can be captured in a SysML activity or sequence diagram. The SysML activity is more suitable for specifying abstract behavior definitions, and the sequence diagram should be used when you need to distinguish between synchronous and asynchronous interactions.

In this example, we chose the SysML activity diagram.

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

1. In the *Use Cases of Vehicle In Use SC* diagram, right-click the shape of the *Provide 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.

i If you refer to the use case diagram, which displays the *Provide Comfortable Temperature* use case, you can see that the shape of that use case is decorated with the rake () icon. This decoration means that the use case contains an internal diagram, the *Provide Comfortable Temperature* activity diagram. Double-clicking the shape of the use case opens that internal diagram.



Step 5. Creating swimlanes and setting allocation mode

Swimlanes in the activity diagram are used to represent the participants of the related system context (see Chapter [System Context](#)). They allow you to establish allocation relationships between the structural and behavioral parts of the model. Use these relationships in the black-box analysis model to specify which participant of the system context is responsible for performing which step of the related use case scenario.

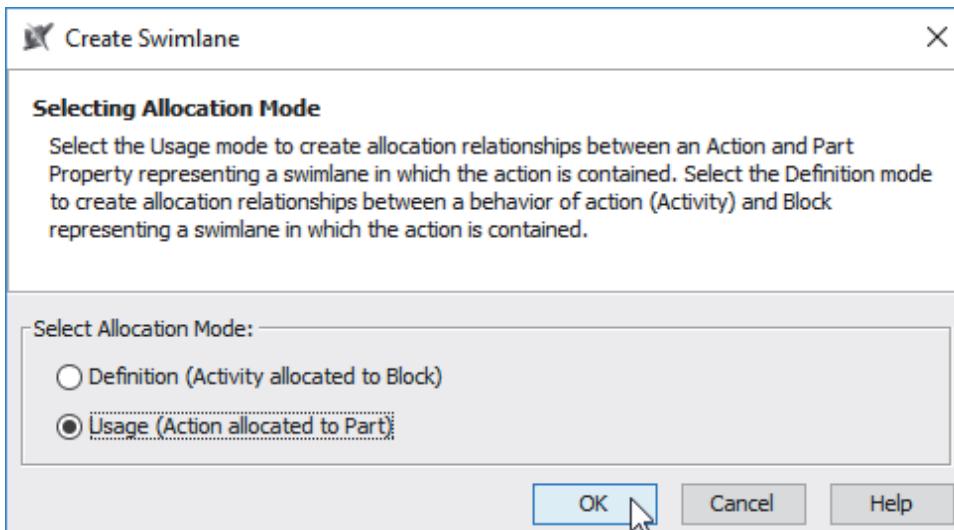
Remember that participants of the system context are captured as its part properties. When you create swimlanes in the activity diagram of the nested use case, the modeling tool automatically recognizes and suggests the part properties of the nesting system context for you to select for representation.

Additionally, you need to enable the allocation to usage mode 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.

Now let's represent the participants of the *Vehicle In Use* system context in the *Provide Comfortable Temperature* use case scenario.

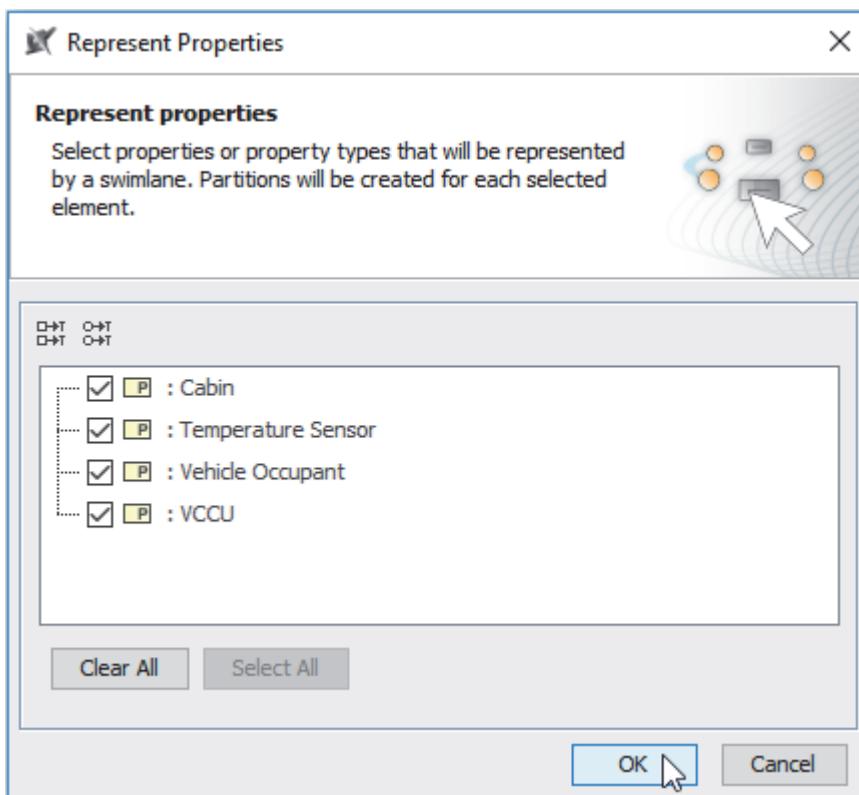
To create swimlanes in the *Provide Comfortable Temperature* activity diagram

1. Open the activity diagram created in [Step 4 of this cell tutorial](#), if not yet opened.
2. Click the **Vertical Swimlanes** button on the activity diagram palette and then click an empty space on that diagram pane.
3. In the open dialog, select **Usage (Action allocated to Part)** to enable the allocation to usage mode and then click **OK**.



i By default, this mode is enabled in both *MagicGrid* templates.

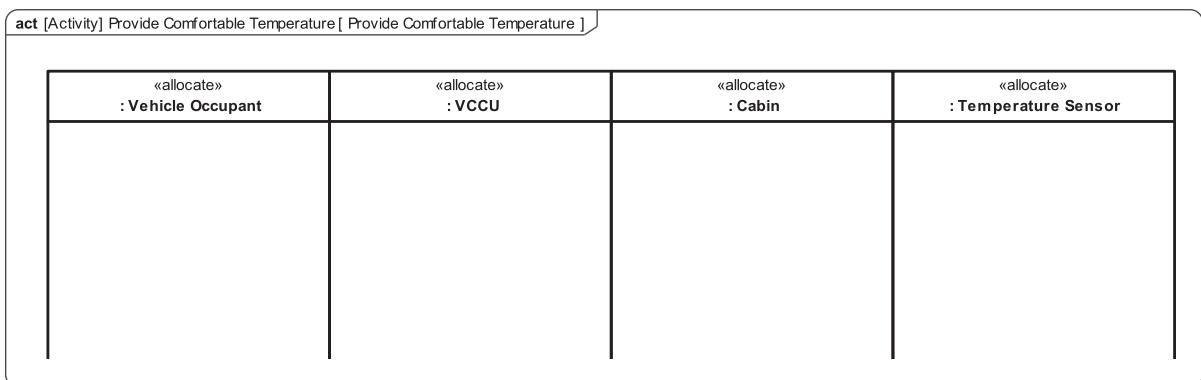
4. In the **Represent Properties** dialog, click **OK** to represent all the participants of the related system context.



i As in this case, the dialog always displays the list of part properties that belong to the related system context. Technically speaking, the related system context is the subject of the use case, which encloses the activity diagram (see [step 3 of this cell tutorial](#)).

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

i Swimlanes are always arranged alphabetically from A to Z, but the buttons **◀** and **▶** (which appear when you select the header of the swimlane) enable you to rearrange them according to your preference in the activity diagram.



When you are finished, you're ready to specify the *Provide Comfortable Temperature* use case scenario.

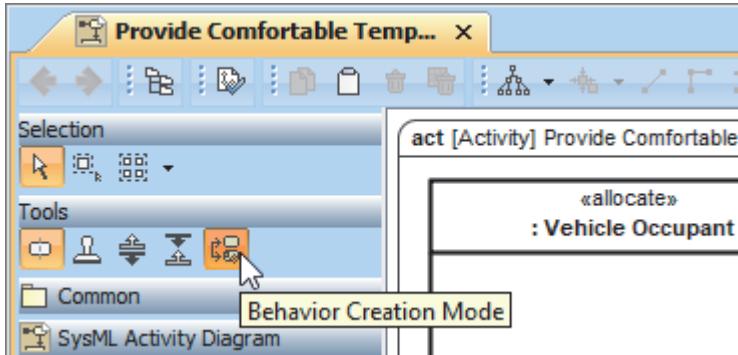
Step 6. Specifying steps of the use case scenario

The steps of the use case scenario can be captured in the activity diagram as call behavior actions. As defined by SysML, a call behavior action may have an assigned behavior, captured in the model as an activity, state machine, or interaction. As an activity, the behavior can be created along with the referencing call behavior action, if the Behavior Creation Mode is enabled in your model. We recommend modeling in this mode for the following reasons:

- It is easier to specify recurrent steps. Any step created in the model once, as an activity assigned to some call behavior action, can then be referred by other call behavior actions in one or more activity diagrams.
- When it is time to decompose a step, there is no need to create the referred activity. It is already created and assigned as a behavior to the call behavior action which captures that step.
- The Activity Decomposition Map does not include call behavior actions. To ensure you have the entire decomposition of the use case steps, you need to have them captured as activities.

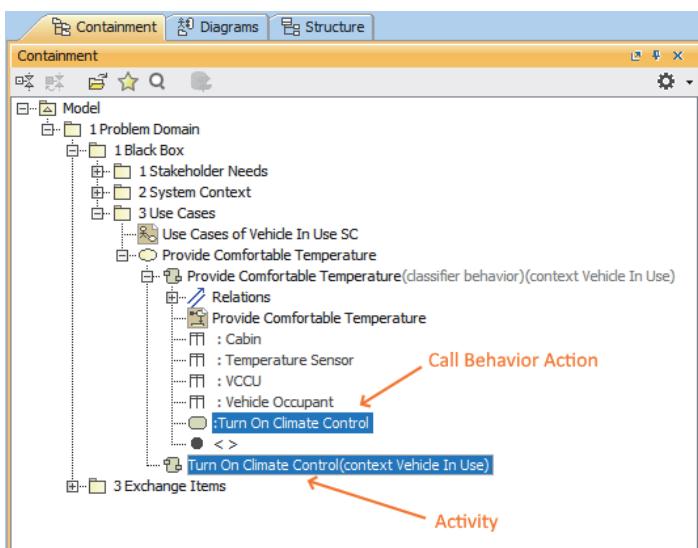
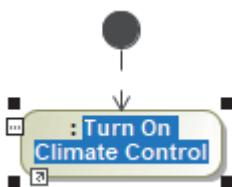
To capture the steps of the main flow of the *Provide Comfortable Temperature* use case scenario

- Be sure the Behavior Creation Mode is enabled in the diagram (see the following figure). Otherwise, activities will not be created and assigned as behaviors to call behavior actions.



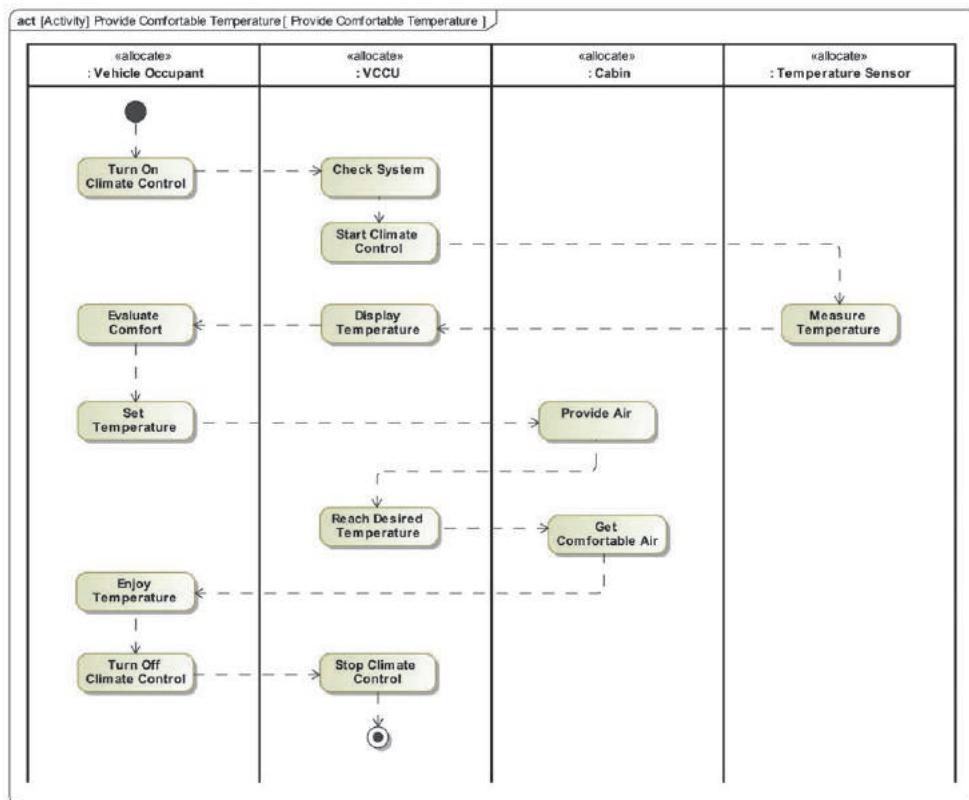
2. Click the **Initial Node** button on the diagram palette and then click an empty place on the diagram pane, within the *:Vehicle Occupant* swimlane. The initial node is created and displayed on the diagram.
3. Click the Control Flow button on the smart manipulator toolbar of the initial node and then click an empty place on the diagram pane in the same swimlane. An unnamed action with a new activity assigned as its behavior is created.
4. Type *Turn on Climate Control* to name the activity and press Enter.

i Be sure to type the name after the colon (""). Otherwise, the name is given to the action, but not to the referred activity.



5. Click the Control Flow button on the smart manipulator toolbar of the newly created action and then click an empty place on the diagram pane, within the *:VCCU* swimlane. An unnamed action with a new activity assigned as its behavior is created.
6. Type *Check System* to name the activity. Remember to type after the colon. Press Enter after you are finished.
7. Repeat steps 5 and 6 as many times as you need to create the actions you see in the following figure.
8. Select the shape of the Stop Climate Control action and click the Control Flow button on its smart manipulator toolbar.

9. Right-click an empty place on the diagram pane and select **Activity Final**. The final node is created and displayed on the diagram.



Now let's split the flow into two: the main flow, when the system can be started after checking, and the alternative flow, when it cannot. To do this, we need to insert a decision node between the *Check System* and *Start Climate Control* actions, and a merge node before the activity final node.

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 *Start Climate Control* actions. The decision node is inserted.

(i) Before inserting the new shape, you can make more space between the existing shapes on the diagram pane. To do this, click the Pusher button on the diagram palette.

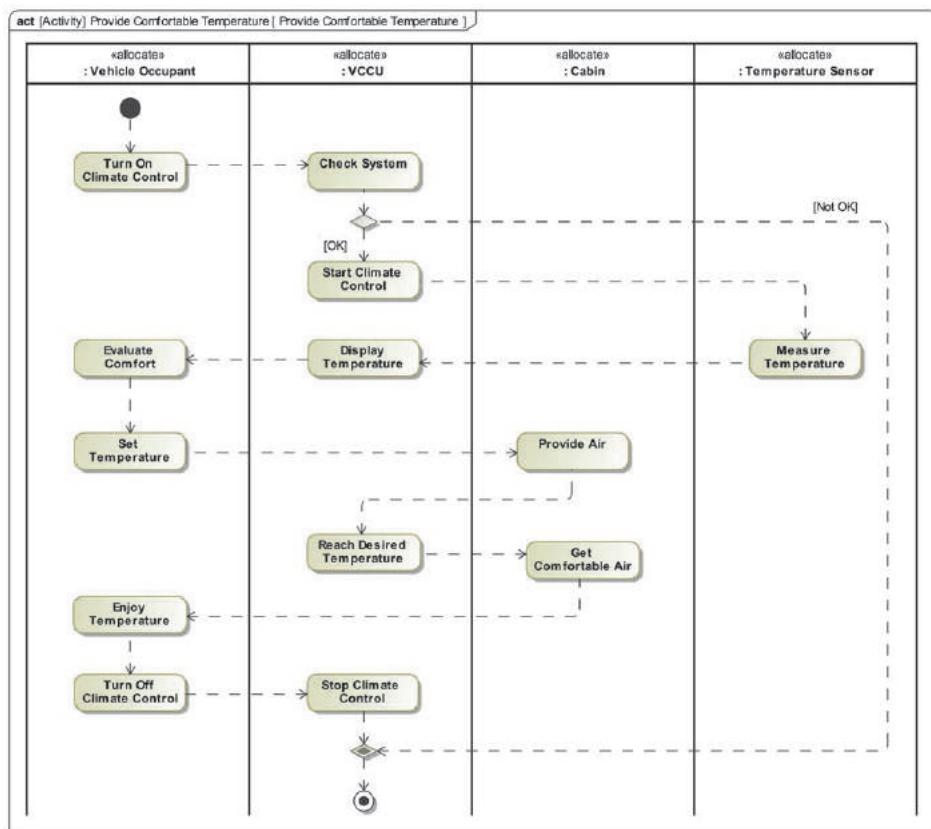
2. Select the control flow between the decision node and the *Start Climate Control* action, type *[OK]*, and press Enter (the closing bracket is added automatically). As a result, the *OK* guard is specified for this control flow and displayed between square brackets on the diagram.

- i** If you open the Specification of the control flow, you can see that the text between the square brackets has been set as the **Guard** property value.

Control Flow	
Name	Provide Comfortable Temperature(classifier behav...
Owner	[1 Problem Domain::1 Black Box::3 Use Cases::P...
Applied Stereotype	[1 Problem Domain::1 Black Box::3 Use Cases::P...
Source	:Start Climate Control [1 Problem Domain::1 Blac...
Target	
Guard	OK
To Do	
Probability	
Rate	
Redefined Edge	

- Click the **Merge** button on the diagram palette and then click the control flow before the activity final node.
- Select the decision node, click the Control Flow button  on its smart manipulator toolbar, then click the merge node. The control flow is created.
- Select that control flow, type *[Not OK]*, and press Enter. The *Not OK* guard is specified for this control flow and displayed between square brackets on the diagram.

After you finish, your *Provide Comfortable Temperature* diagram should look very similar to the one in the following figure.

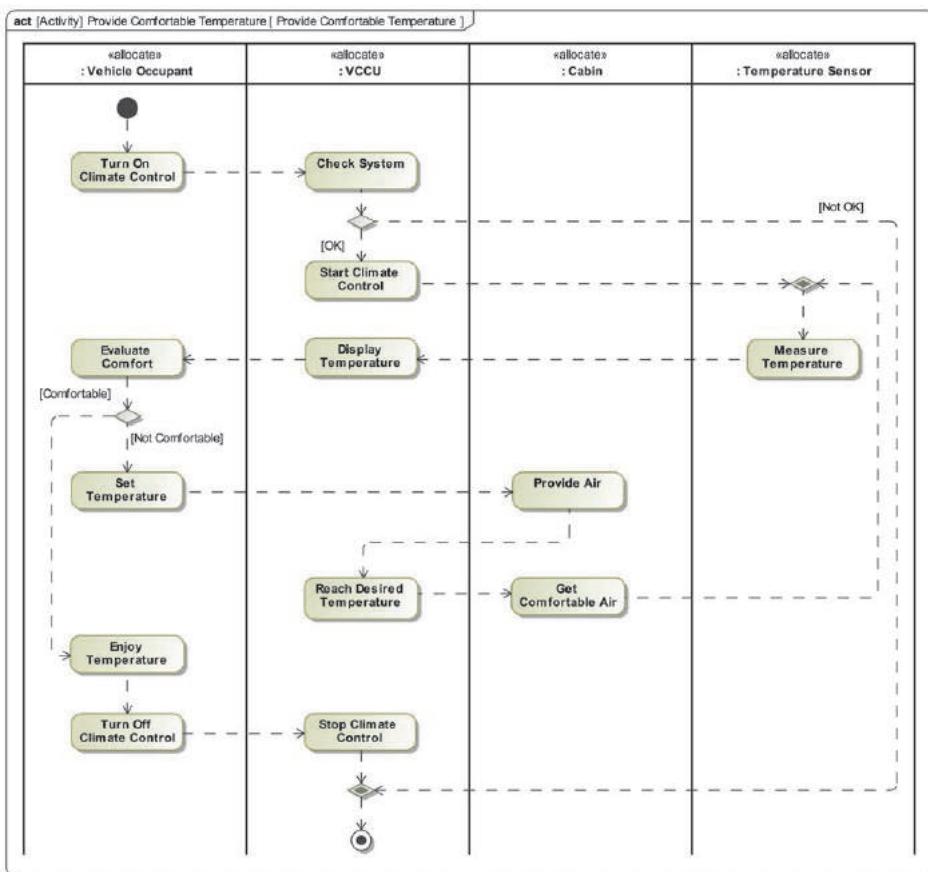


There is one more alternative flow that must be specified in the scenario, which says the vehicle occupant can either set the desired temperature or enjoy the ambient temperature after they evaluate the comfort in the cabin. For this, you need to insert another pair of decision and merge nodes in the diagram.

To update the use case scenario with another alternative flow

1. Click the **Decision** button on the diagram palette and then click the control flow between the *Evaluate Comfort* and *Set Temperature* actions. The decision node is inserted.
2. Select the control flow between the decision node and the *Set Temperature* action, type *[Not Comfortable]*, and press Enter. The *Not Comfortable* guard is specified for this control flow and displayed between square brackets on the diagram.
3. Click the Control Flow button  on the smart manipulator toolbar of the newly created decision node and then select the *Enjoy Temperature* action. A new control flow is created.
4. Immediately type *[Comfortable]*. The *Comfortable* guard is specified for this control flow and displayed between square brackets on the diagram.
5. Click the **Merge** button on the diagram palette and then click the control flow between the *Start Climate Control* and *Measure Temperature* actions.
6. Select the control flow between the *Get Comfortable Air* and *Enjoy Temperature* actions, click its arrow end, and then drag it to the newly created merge node. The target node of the control flow changes.

After you finish, your *Provide Comfortable Temperature* diagram should look very similar to the one in the following figure.



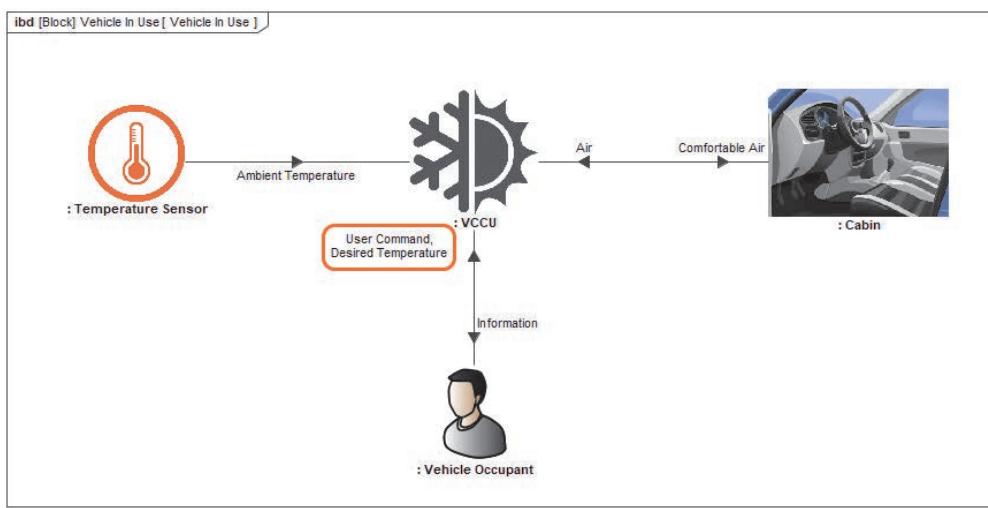
The simulation engine of the **modeling tool**, based on the OMG® Executable UML (fUML)™ standard, enables you to validate the specified behavior and see if your model is correct. To start the simulation session, click the Run button  on the *Provide Comfortable Temperature* activity diagram toolbar. Then in the **Simulation** panel, click the Start button  to start the model execution.

So far, we have used control flows to connect actions. In the next step we will specify what data, matter, or energy flows between actions.

Step 7. Adding item flows to the use case scenario

Now we will update the *Provide Comfortable Temperature* use case scenario with item flows, by synchronizing them from the *Vehicle In Use* ibd created during the system context analysis (see Chapter [System Context](#)). This helps to identify and eliminate logical inconsistencies across the structural and behavioral parts in the problem domain model, such as the existence of a questionable item flow that does not appear in any use case scenario, or identifying the need for a new item flow.

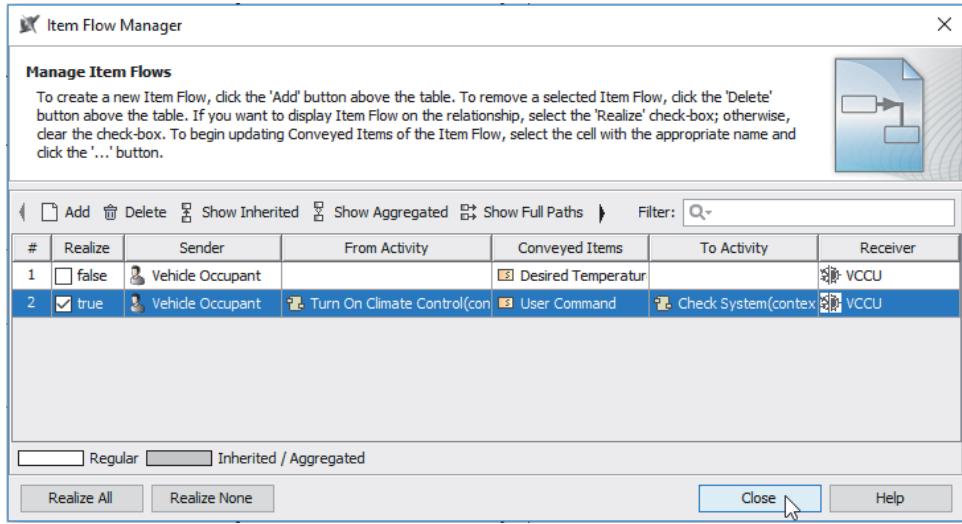
Before adding item flows to the activity diagram which captures the *Provide Comfortable Temperature* use case scenario, we need to look at the *Vehicle In Use* ibd first. Let's start from the interaction between the *:Vehicle Occupant* and the *:VCCU* part properties. We see that the vehicle occupant provides control commands to and sets the desired temperature for the VCCU (see the highlighted fragment of the diagram below).



The item flows conveying these signals must be realized between these part properties in the use case scenario, as well. Remember that part properties specified in the *Vehicle In Use* ibd are represented by swimlanes in the *Provide Comfortable Temperature* activity diagram. Item flows should be conveyed by object flows, not control flows. Therefore, we need to convert relevant control flows to object flows.

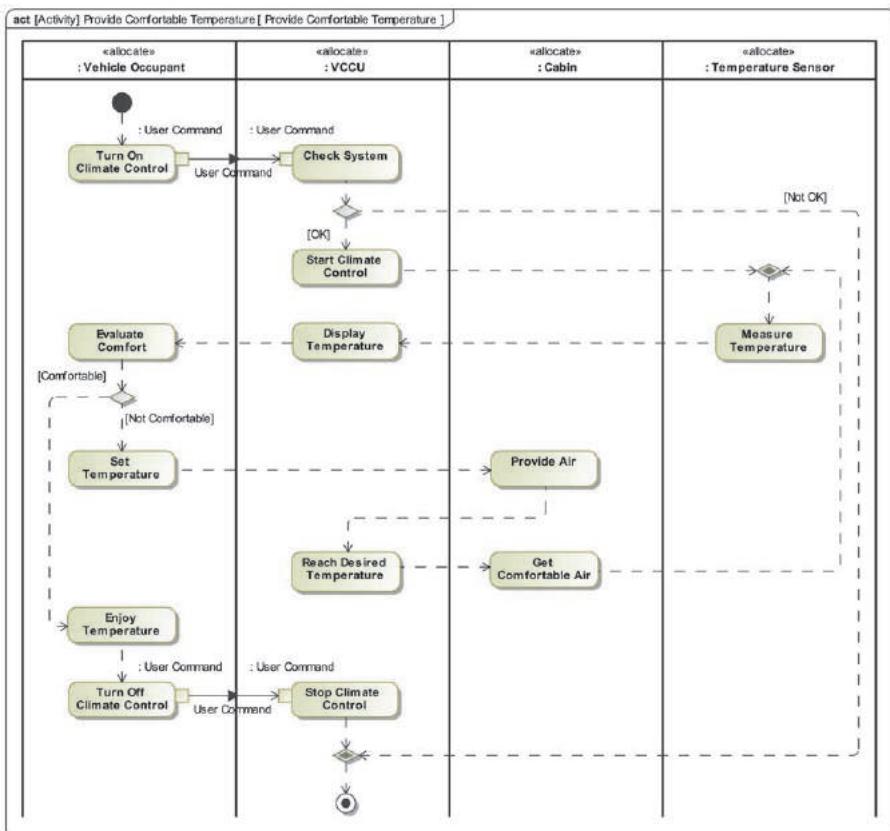
To realize item flows conveying the *User Command* signal between the actions allocated to the *:Vehicle Occupant* and the *:VCCU* part properties

1. Select the control flow between the *Turn On Climate Control* and *Check System* actions.
2. Right-click the selection and choose **Refactor > Convert To > Object Flow**. The control flow turns to the object flow, and pins are created for both actions.
3. In the Model Browser, under the *3 Exchange Items* package, select the *User Command* signal.
4. Drag the signal to the output pin of the *Turn On Climate Control* action. The *User Command* type is set as the type for the output pin.
5. Drag the signal to the input pin of the *Check System* actions. The *User Command* type is set as the type for the input pin.
6. Select the object flow and click Item Flow Manager button from its smart manipulator toolbar.
7. In the open dialog, select the item flow that conveys the *User Command* signal (see the following figure). The item flow is realized between the relevant actions.



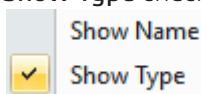
8. Click **Close** to close the dialog.
9. Repeat the steps 1 to 8 for the *Turn Off Climate Control* and *Stop Climate Control* actions.

When you finish, your diagram should look the same as in the figure below.



- i** You may notice that pins display their types only. For this, do the following:

1. Press down the Alt key and select any input pin. All input pins are selected on the diagram.
2. Right-click the selection, click **Show Name** to clear the selection, and then click to select the **Show Type** check box.

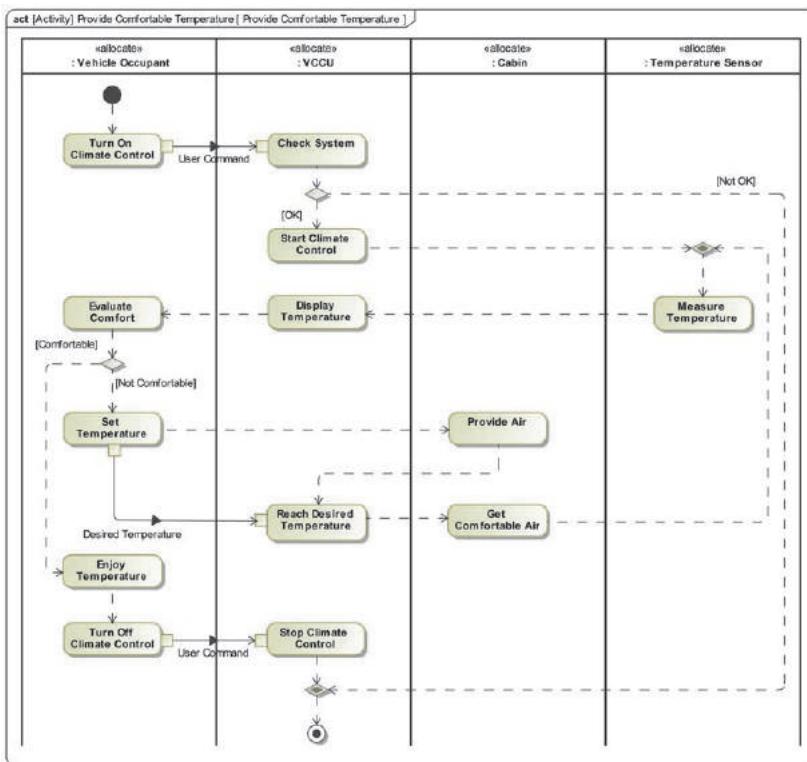


3. Repeat the steps above to hide the output pin types.

To realize item flows conveying the *Desired Temperature* signal between the actions allocated to the *:Vehicle Occupant* and the *:VCCU* part properties

1. Select the *Set Temperature* action and click the Object Flow button  on its smart manipulator toolbar.
2. Click the *Reach Desired Temperature* action. The object flow with pins for both actions is created.
3. In the Model Browser, under the *3 Exchange Items* package, select the *Desired Temperature* signal.
4. Drag the signal to the output pin of the *Set Temperature* action. The *Desired Temperature* type is set as the type for the output pin.
5. Drag the signal to the input pin of the *Reach Desired Temperature* actions. The *Desired Temperature* type is set as the type for the input pin.
6. Select the object flow and click Item Flow Manager button  on its smart manipulator toolbar.
7. In the open dialog, select the item flow that conveys the *Desired Temperature* signal. The signal becomes the conveyed item of the object flow.
8. Click **Close** to close the dialog.

When you finish, your diagram should look the same as in the figure below.



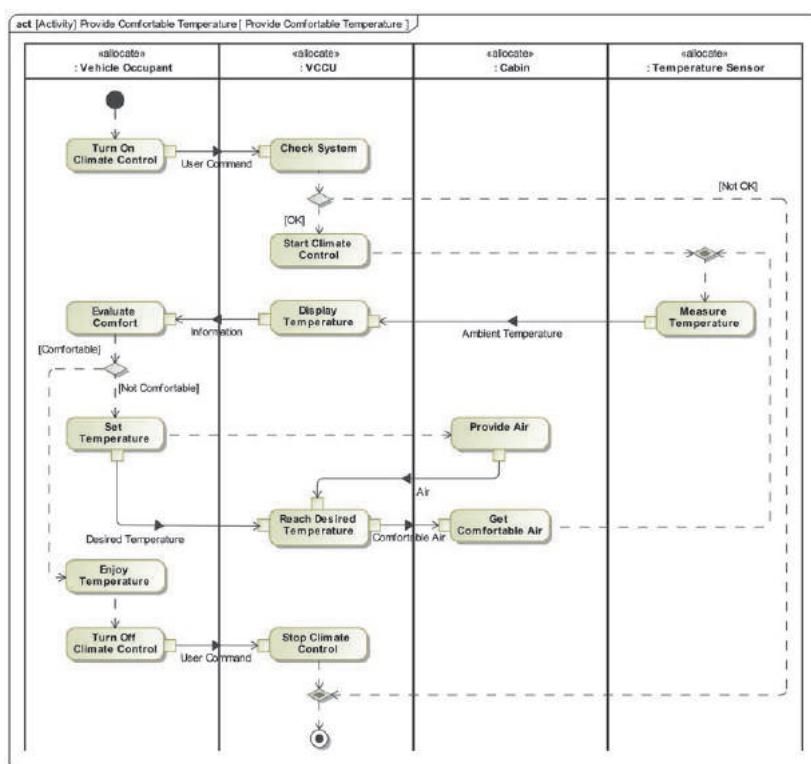
i You may notice that pins don't display their types, and only item flows are shown on the diagram. This makes the diagram less crowded and easier to read. If you want to get rid of pin types, do the following:

1. Press down the Alt key and select any input pin. All input pins are selected on the diagram.
2. Right-click the selection and click **Show Type** to clear the selection.
3. Repeat the steps above to hide the output pin types.

Use the same method to specify the following:

- The *Ambient Temperature* signal flowing between the *Measure Temperature* and *Display Temperature* actions. Remember the flow between the *:Temperature Sensor* and *:VCCU* part properties of the *Vehicle In Use* system context.
- The *Air* block being conveyed from the *Provide Air* action to the *Reach Desired Temperature* action. Remember the flow between the *:Cabin* and *:VCCU* part properties of the *Vehicle In Use* system context.
- The *Comfortable Air* block being conveyed from the *Reach Desired Temperature* action to the *Get Comfortable Air* action. Remember the flow between the *:VCCU* and *:Cabin* part properties of the *Vehicle In Use* system context.
- The *Information* signal flowing between the *Display Temperature* and *Evaluate Comfort* actions. Remember the flow between the *:VCCU* and *:Vehicle Occupant* part properties of the *Vehicle In Use* system context.

After you finish, your diagram should look similar to the one below. You may notice that all the item flows here are the same as in the *Vehicle In Use* ibd. Always remember that a precise and complete model requires item flow synchronization between system structure and behavior. For this reason, all exchange items identified in the structural model must be considered and included in the behavioral model as well.



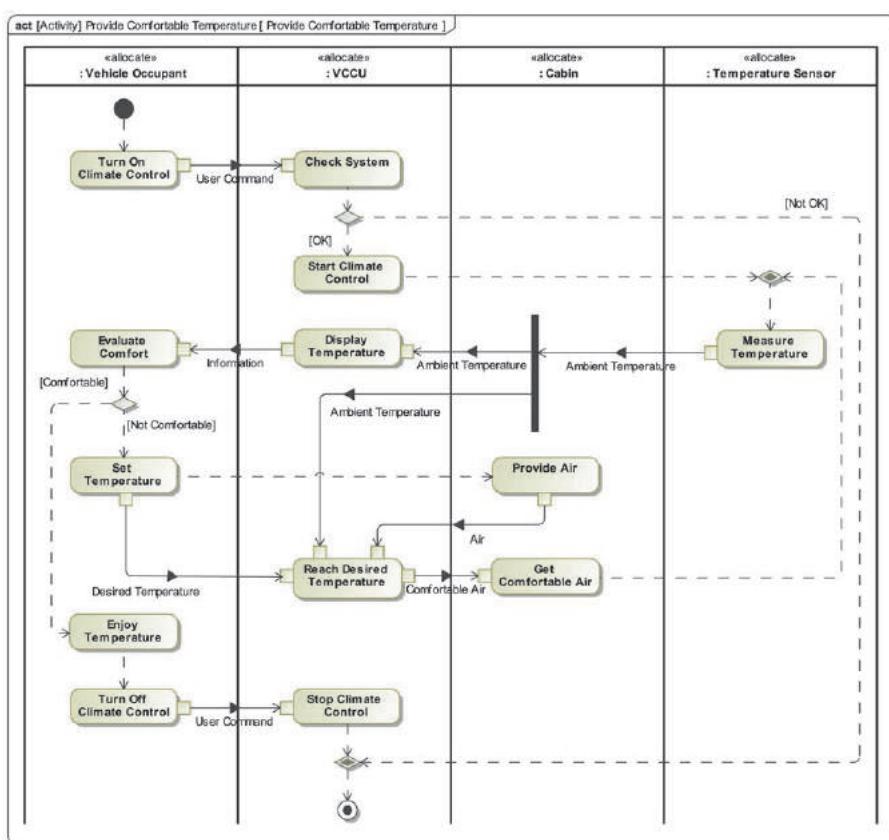
Now you can initiate and run the simulation session to execute the model and validate the specified behavior.

Step 8. Supplementing the use case scenario with a parallel flow

The *Provide Comfortable Temperature* scenario still needs a few updates to be fully completed. We also need to specify that the *Ambient Temperature* signal is obtained by the *Reach Desired Temperature* action. Therefore, we have to split the flow going out of the *Measure Temperature* action into two. For this, the fork node can be utilized.

To split the object flow going out of the *Measure Temperature* action into two

1. Click the small black triangle next to the **Fork Horizontal** button on the diagram palette and then select **Fork Vertical**.
2. Click the object flow between the *Measure Temperature* and *Display Temperature* actions. The vertical fork node is inserted.
3. Click the vertical fork node and then click the Object Flow button  on its smart manipulator toolbar. Select the *Reach Desired Temperature* action. The object flow and input pin for that action are created. The input pin is automatically typed by the *Ambient Temperature* signal. You do not need to change anything.
4. Select the object flow and click the Item Flow Manager button  from its smart manipulator toolbar.
5. In the open dialog, click to select the single item flow that conveys the *Ambient Temperature* signal. The signal becomes the conveyed item of the object flow.
6. Click **Close** to close the dialog.



The use case scenario can now be considered as complete. You can initiate and run the simulation session again, to execute the model and validate the specified behavior.

Use Cases done. What's next?

Having created one or more use case scenarios, you are ready for functional analysis. Steps that are allocated to the **Sol** (in SysML terms, they appear in the swimlane that represents the **Sol**) can be decomposed to specify the internal behavior expected from the **Sol** (see Chapter [Functional Analysis](#)).

Measures of Effectiveness

What is it?

In this cell, non-functional stakeholder needs are refined with measures of effectiveness (MoEs), which capture quantifiable characteristics of the **Sol** in a 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 high level key performance indicators that would be automatically checked within the **solution domain** model: with MoEs, you can specify design constraints that can be used to verify the system design.

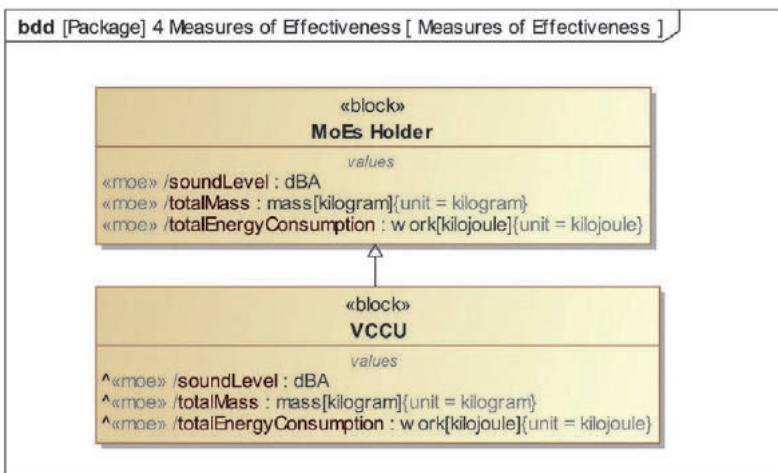
The same set of MoEs can be used in different models to specify numerical characteristics of other systems of interest.

Who is responsible?

MoEs can be identified by the Systems Analyst or Systems Engineer.

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 **Sol**. In real-world projects, blocks with reusable sets of MoEs can be stored in external models, also known as libraries. MoEs are captured as value properties with «moe» stereotypes applied. The **Sol** block 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.

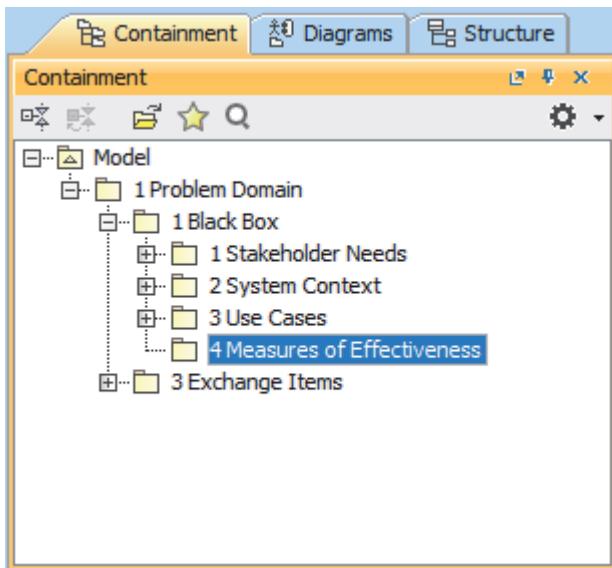


Tutorial

- Step 1. Organizing the model for MoEs
- Step 2. Creating a block for capturing MoEs
- Step 3. Capturing MoEs for the Sol

Step 1. Organizing the model for MoEs

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



To organize the model for measures of effectiveness

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 Measures of Effectiveness* to specify the name of the new package and press Enter.

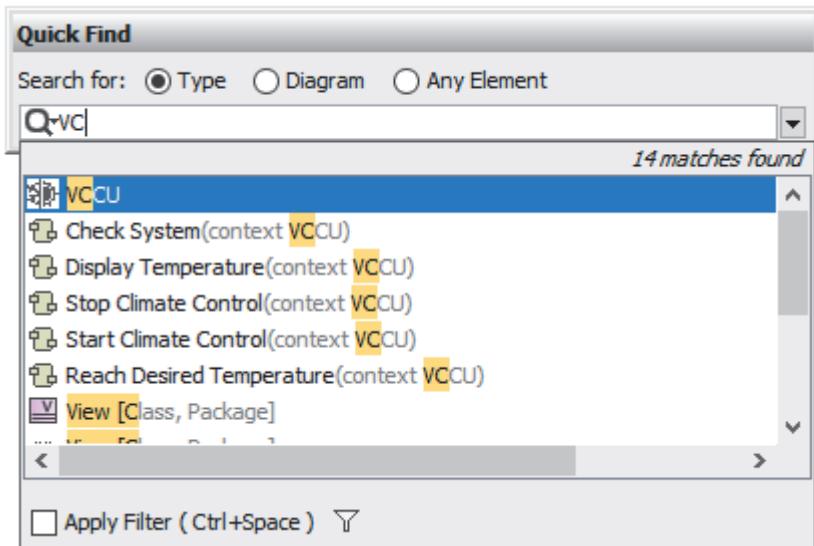
Step 2. Creating a block for capturing MoEs

To capture 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 *VCCU* block. Remember that if it was a real-world project, the MoEs holder could be created in an external model, also known as a library.

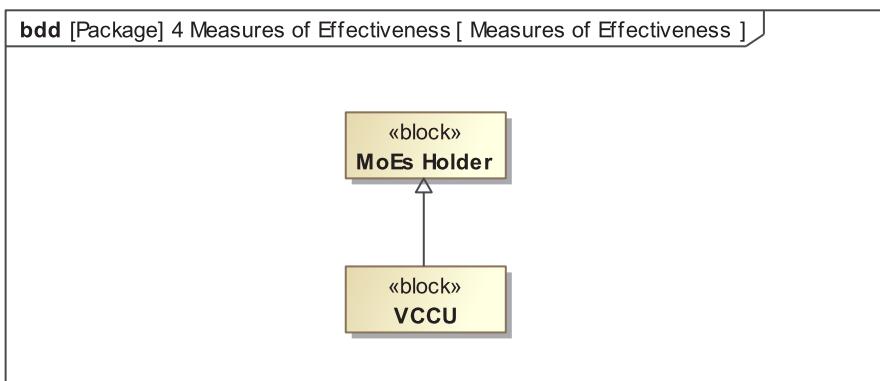
To create a diagram and a block for capturing MoEs

1. Right-click the *4 Measures 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-click Enter. The diagram is created.
- i** Note that the diagram is named after the package where it is stored. This name also suits the diagram. 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 *VCCU* block, using the quick find capability:
- Press Ctrl + Alt + F. The **Quick Find** dialog opens.
 - Type *VC*.
 - When you see the *VCCU* block selected in the search results list (see the following figure), press Enter. The *VCCU* block is selected in the Model Browser.



6. Drag the *VCCU* block to the diagram pane. The shape of the *VCCU* block appears on the diagram.
 7. Click the Generalization button on the smart manipulator toolbar of the *MoEs Holder* block and then click the shape of the *VCCU* block. The *MoEs Holder* block becomes the super-type of the *VCCU* block.



Step 3. Capturing MoEs for the SoI

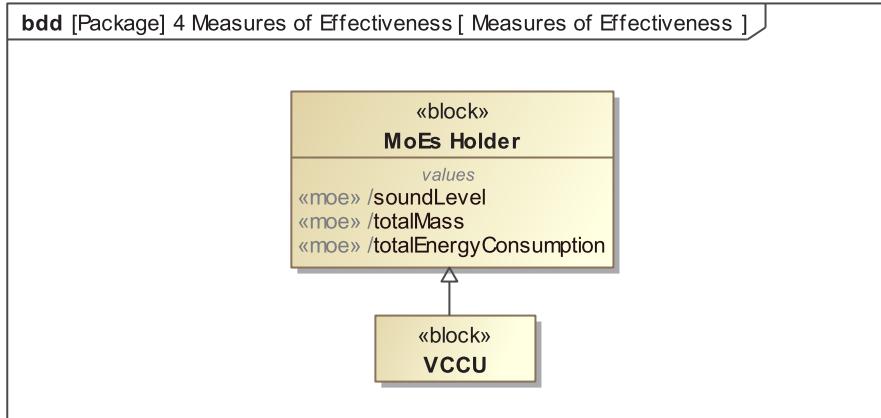
Now we will analyze non-functional stakeholder needs *SN-1.1.1*, *SN-1.1.4*, and *SN-1.2.1*. They prompt you to create the MoEs for the sound level, total mass, and total energy consumption. Let's capture these MoEs in your model.

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 *soundLevel* to specify the name of the new value property and press Enter.

You can type "/" in front of the property name, for example, */soundLevel*, to indicate that it is derivable. Otherwise, to set the **Is Derived** property value to *true*, you need to right-click the property and then select the **Is Derived** check box.

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 to the *soundLevel* value property.
6. Repeat the steps above to capture the *totalMass* and *totalEnergyConsumption* MoEs.

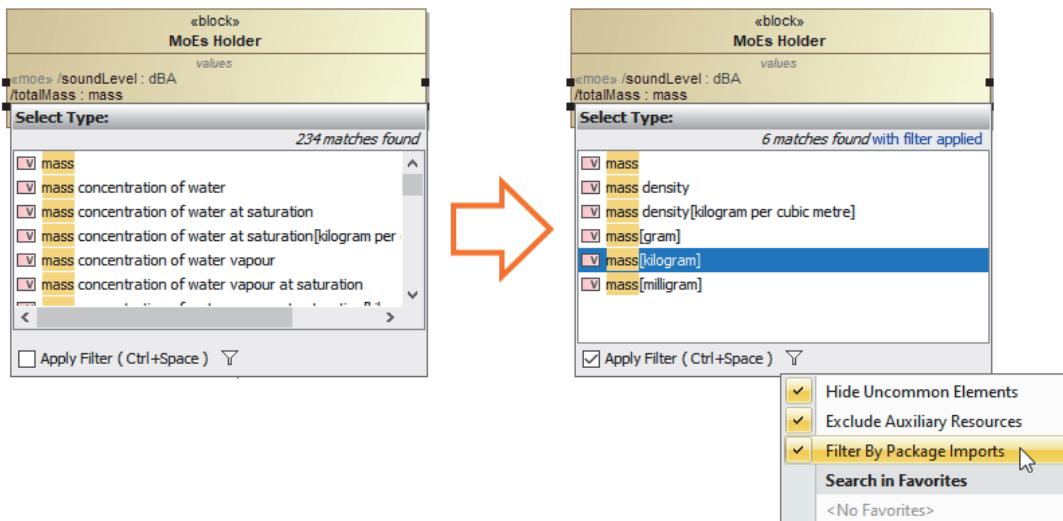


MoEs may have units, which can be captured in the model as value types. The majority of the commonly used value types are stored in the ISO 80000 library of units, which is provided by the [modeling tool](#).

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 *totalMass* 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.

- i** The list of possible value types can be narrowed down. For this, click to select the **Apply Filter** check box and then make sure the **Filter By Package Imports** check box (under the Filter Options button ) is selected.



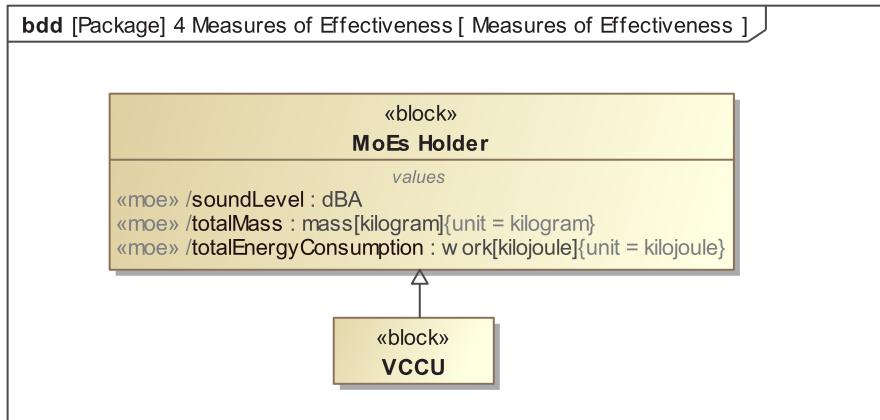
By default, value types are filtered to show basic units only. However, the list may not include the required value types, and you may need to extend it. For this, the *Basic Units* matrix should be updated. For more information, refer to the latest documentation of the [modeling tool](#).

- Repeat steps 2 and 3 to find *work[megajoules]* and set it as the type for the *totalEnergyConsumption* value property.

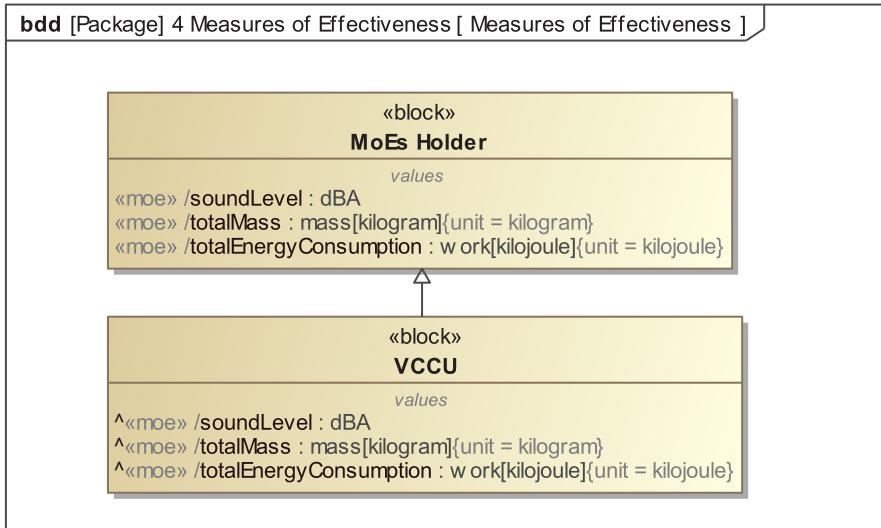
Custom units can be created manually.

To set a custom unit as MoE type

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



If you want to display MoEs on the shape of the *VCCU* block (see the following figure), you need to update the symbol properties of that block. For this, right-click the *VCCU* block, select **Symbol Properties**, and then in the open dialog, find the **Show Inherited** property (make sure the **All** mode is switched on in the dialog, or you will not be able to find it). Once you do this, set the property value to *true*.

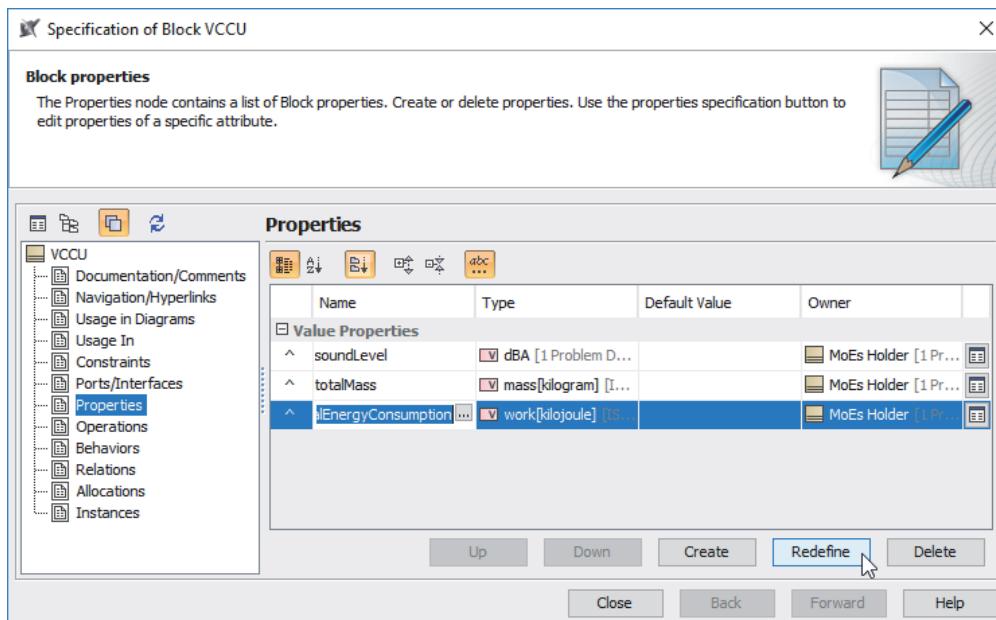


- i** The set of MoEs captured as the *MoEs Holder* block can later be used in different models to assign the same quantifiable characteristics to other systems of interest.

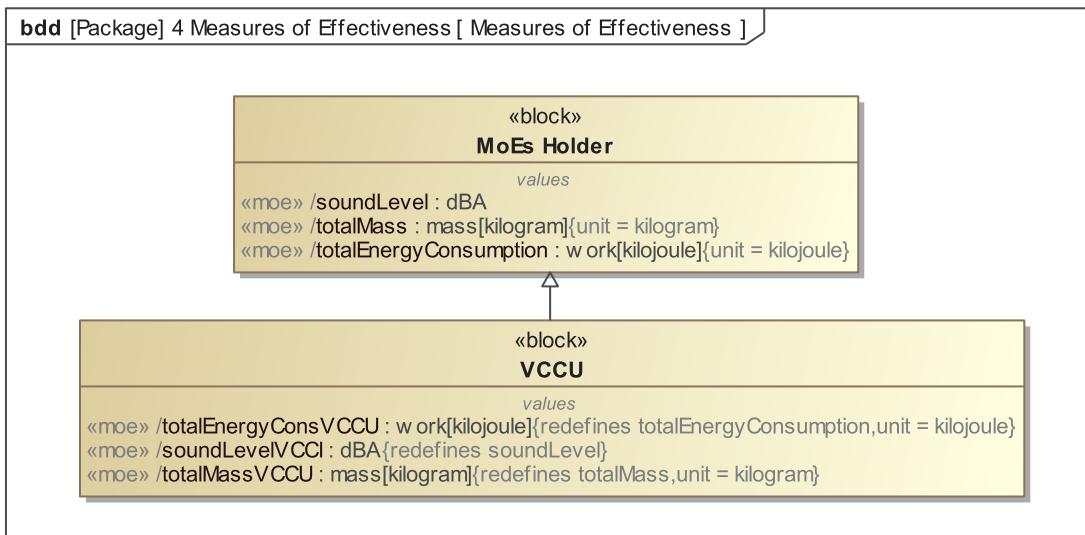
If you want to make changes to the inherited MoEs of the **Sol**, you need to redefine them. Let's say you want to change their names to make them more specific.

To redefine the inherited value properties

1. Double-click the **VCCU** block to open its Specification.
2. On the left side of the dialog, click **Properties**.
3. On the right side, select the *totalEnergyConsumption* value property and click the **Redefine** button below.



4. In the **Name** cell, type the new name of the property (for example, *totalEnergyConsVCCU*).
5. Press Enter to confirm the changes.
6. Repeat steps 3 to 5 to redefine other MoEs, as shown in the following figure.



MoEs done. What's next?

Now that you have measures of effectiveness, you can specify design constraints, which can be used to verify whether the system design is correct or not (see Chapter [System Parameters](#)).

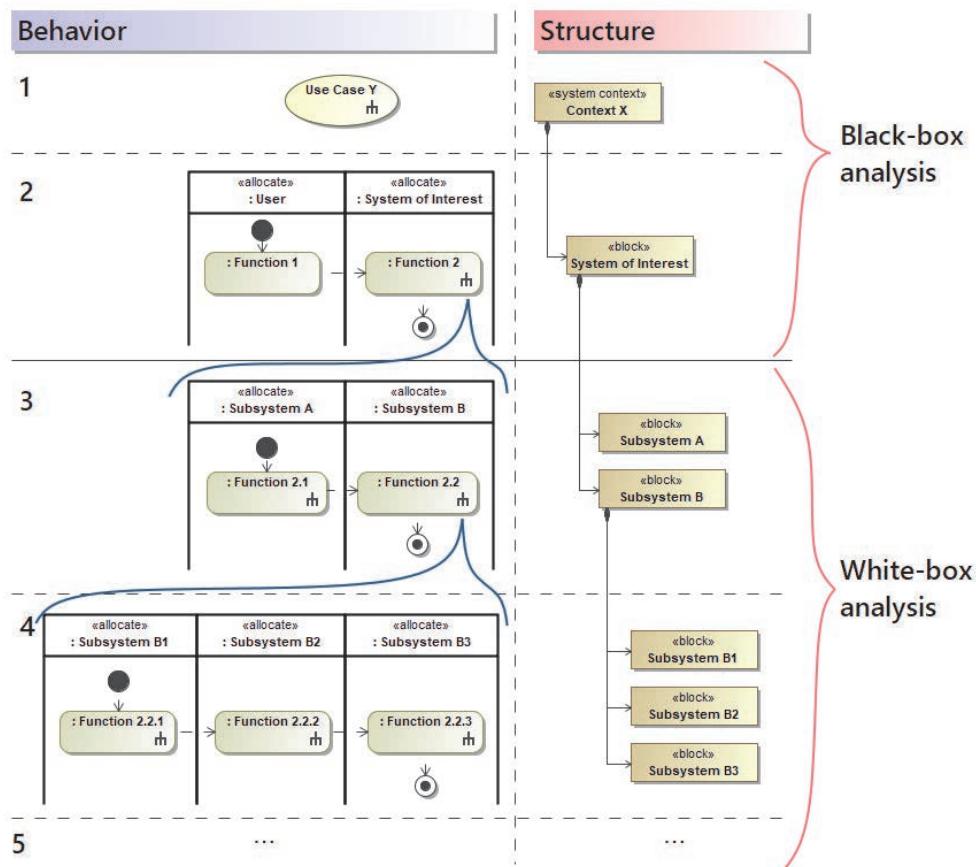
White-box perspective

Once you have finished the problem definition from the black-box perspective (i.e., the operational analysis of the **Sol** is completed), you can switch to analyzing the **Sol** from the white-box perspective. This helps to understand the expected behavior of the **Sol** before producing its logical solution architecture.

In the second phase of the problem domain definition, you should perform deeper analysis of system functions in order to identify the conceptual subsystems (referred to as functional blocks in some methodologies) of the **Sol**. Conceptual subsystems are the first step towards the logical solution architecture of the **Sol**.

Deeper analysis of system functions is called functional analysis. As the following figure shows, every function performed by the **Sol**, as identified in the **Use Case** cell (layer 2), is decomposed in the **Functional Analysis** cell (layer 3). Functions of the detailed behavior are then allocated to conceptual subsystems (aka functional blocks) to convey that each conceptual subsystem (aka functional block) is responsible for performing one or more functions.

The functional decomposition process can have as many iterations as needed to achieve the relevant granularity of the problem domain definition. The following figure shows that functions at each layer of detail can be decomposed into even more detailed behavior. Conceptual subsystems are accordingly decomposed into more elementary structures. It is important to note that the granularity of system behavior and structure must be consistent in each level of detail.



After functional subsystems are identified, interactions between them can be specified. Every subsystem can have its own quantitative characteristics (i.e., measures of effectiveness). The functions of the **Sol**, its

functional subsystems, and [MoEs](#) together establish the basis for specifying system requirements (see Chapter [System Requirements](#)).

		Pillar				
Domain	Problem	Requirements	Structure	Behavior	Parameters	Safety & Reliability
		Stakeholder Needs	System Context	Use Cases	Measures of Effectiveness (MoEs)	Conceptual and Functional Failure Mode & Effects Analysis (FMEA)
		Conceptual Subsystems	Functional Analysis	MoEs for Subsystems	Conceptual Subsystems FMEA	
	Solution	System Requirements	System Structure	System Behavior	System Parameters	System Safety & Reliability (S&R)
		Subsystem Requirements	Subsystem Structure	Subsystem Behavior	Subsystem Parameters	Subsystem S&R
		Component Requirements	Component Structure	Component Behavior	Component Parameters	Component S&R
	Implementation	Implementation Requirements				

Read the following pages to learn how to create the problem domain from the white-box perspective.

Functional Analysis

What is it?

In this cell, you should more deeply analyze the expected system behavior by decomposing every function identified during the black-box analysis (see Chapter [Use Cases](#)) and performed by the [Sol](#).

Decomposing each function into more detailed system behavior gives stimulus to identify the conceptual subsystems (aka functional blocks) that are responsible for performing these functions. Conceptual subsystems are captured in the next cell of the white-box analysis (see Chapter [Conceptual Subsystems](#)).

It is important to note that with 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.

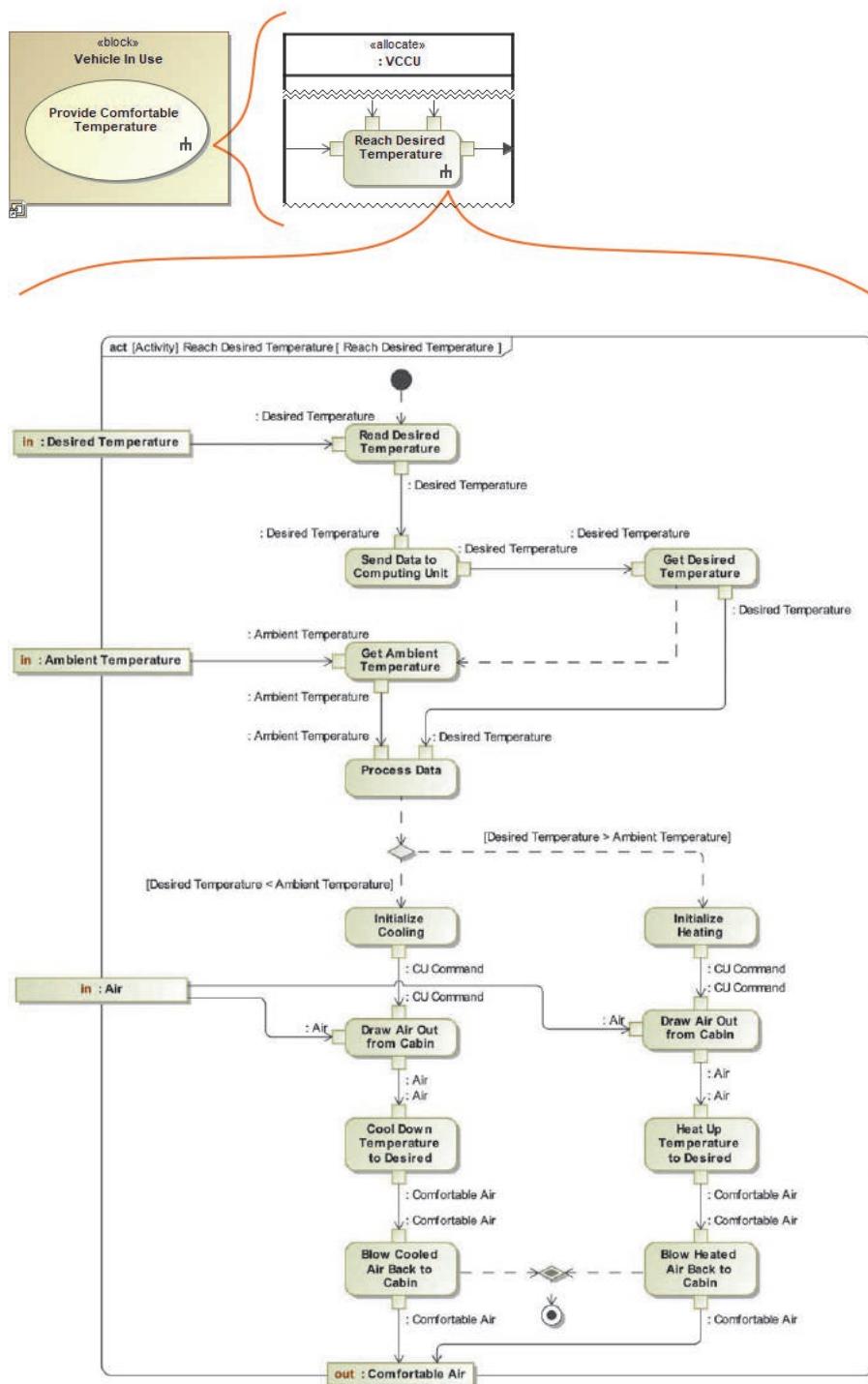
The result of the Functional Analysis cell is the model of the expected system behavior. An activity decomposition map starting from the high-level goals of the [Sol](#) and ending with the expected functions of it can be extracted from that model.

Who is responsible?

As a continuation of the use case analysis, functional analysis can be performed by the Systems Analyst or Systems Engineer.

How to model?

Functional analysis is a continuity of use case scenario refinements using SysML activity diagrams. A new SysML activity diagram should be created for every function allocated to **Sol** (see Chapter [Use Cases](#)). Though there are two or even more swimlane partitions, you should choose only those functions that are nested under the partition representing the usage of block that captures your **Sol**. The following figure shows the white-box scenario of the *Reach Desired Temperature* function (which is a part of the *Provide Comfortable Temperature* use case scenario) performed by the **VCCU Sol**.

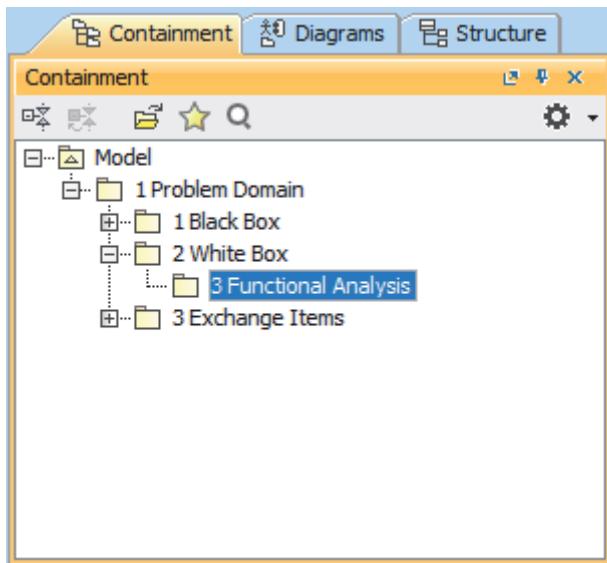


Tutorial

- Step 1. Organizing the model for functional analysis
- Step 2. Creating an activity diagram to decompose a function
- Step 3. Specifying the white-box scenario

Step 1. Organizing the model for functional analysis

According to the design of the MagicGrid framework, model artifacts that capture decomposed system functions should be stored under the structure of packages, as 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 the functional analysis

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 *3 Functional Analysis*.

Step 2. Creating an activity diagram to decompose a function

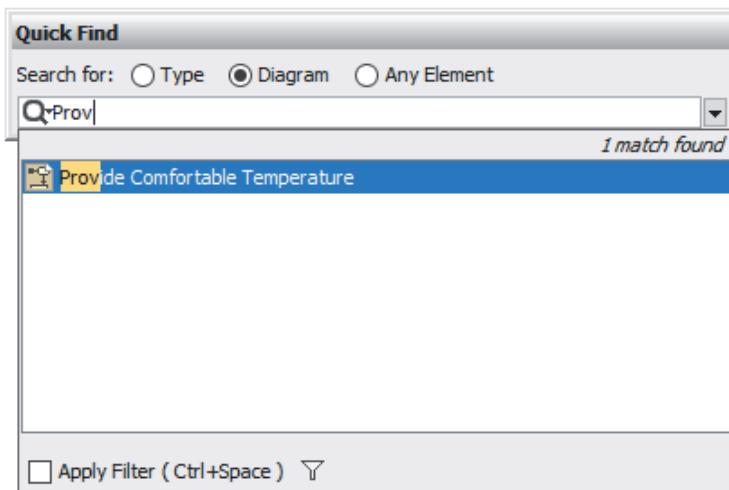
Let's say you want to decompose the function captured in your model as the *Reach Desired Temperature* activity. You will need to create a SysML activity diagram inside that activity.

Additionally, you need to change the location of the *Reach Desired Temperature* activity in your model by moving it to the *3 Functional Analysis* package, a sub-package of the *2 White Box* package (created in the previous step of this cell tutorial).

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

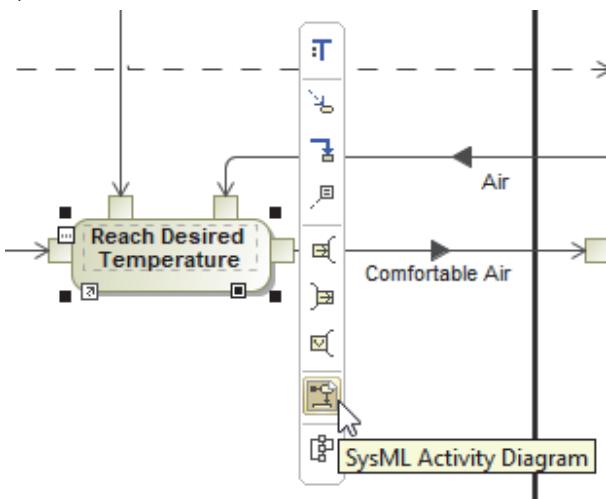
1. Open the *Provide Comfortable Temperature* activity diagram:

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

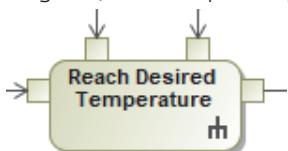


2. Select the shape of the *Reach Desired Temperature* action and click the SysML Activity Diagram button

- on its smart manipulator toolbar (see the following figure). A newly created activity diagram opens.

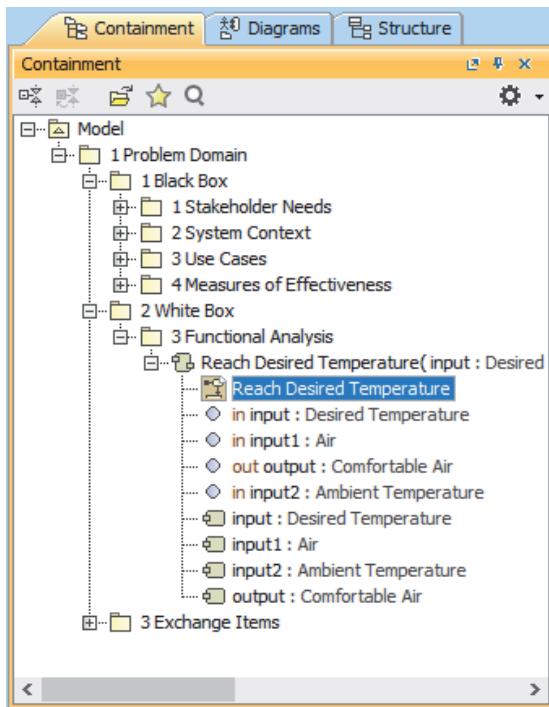


If you go back to the *Provide Comfortable Temperature* activity diagram (simply by clicking on the open diagram toolbar), you can see that the shape of the *Reach Desired Temperature* action is decorated with the rake () icon, indicating that the action contains an internal diagram, which opens by double-clicking the shape of that action.

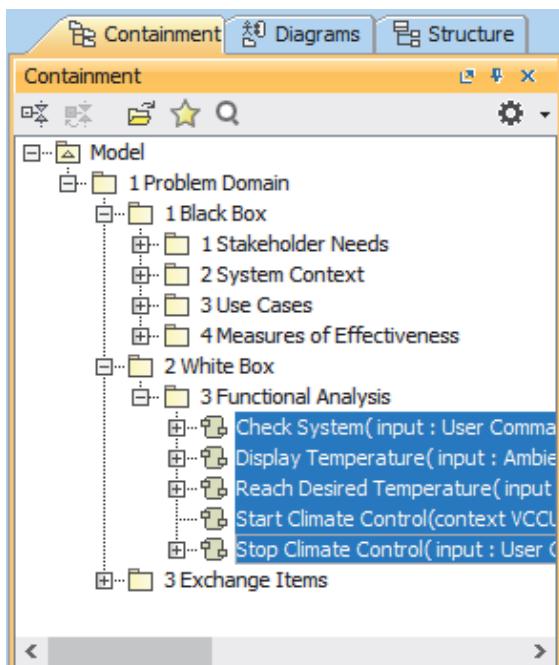


3. In the Model Browser, select the *Reach Desired Temperature* activity:
 - a. Right-click the shape of the *Reach Desired Temperature* action on the diagram pane.
 - b. Select **Go To > Behavior Reach Desired Temperature**. The activity is selected in the Model Browser.

4. Drag the *Reach Desired Temperature* activity to the package *3 Functional Analysis* (within the *2 White Box* package). The related activity diagram is moved in together.



When you finish the white-box analysis in a real-world project, all the activities that capture the functions of the **Sol** (they appear within the *:VCCU* swimlane in the *Provide Comfortable Temperature* activity diagram) must be stored in the *3 Functional Analysis* package.



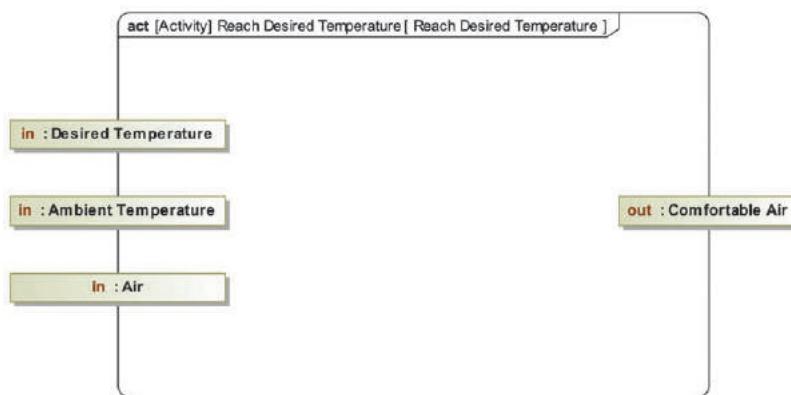
Step 3. Specifying the white-box scenario

Now we will specify the white-box scenario of the function captured as the *Reach Desired Temperature* activity.

The activity diagram has a few inputs and an output. In terms of SysML, these are activity parameter nodes and they correspond to the input and output pins of the *Reach Desired Temperature* action in the *Provide Comfortable Temperature* activity diagram. Inputs and outputs are a good starting point and should be considered when specifying the scenario. Additionally, you can remove their names to remove unnecessary information and have more space on the diagram pane.

To remove the name of the activity parameter node

1. Click the node once. When it is selected, click its name. The name is selected.
2. Press the Delete button on the keyboard.



This time before starting to specify the scenario, you don't need to create any swimlanes in the diagram. You can do this later, as soon as you have conceptual subsystems (aka functional blocks) captured in your model.

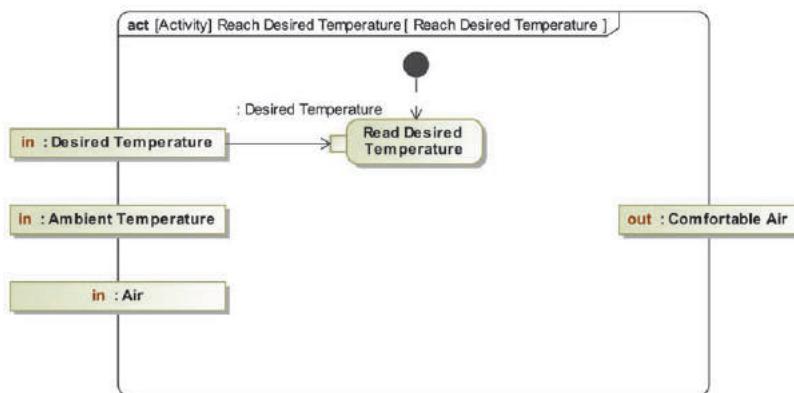
To specify the white-box scenario of the *Reach Required Temperature* activity

1. Be sure the Behavior Creation Mode is enabled in the diagram. If it is not enabled, activities will not be created and assigned as behaviors to call behavior actions. For this, click the Behavior Creation Mode button .
2. Click the **Initial Node** button on the diagram palette and then click an empty place on the diagram pane. The initial node is created.
3. Click the Control Flow button  on the smart manipulator of the initial node shape, and then click an empty place on the diagram pane. An unnamed action with a new activity assigned as its behavior is created.
4. Type *Read Desired Temperature* and press Enter.

(i) Be sure you type the name after the colon (":"), or the name will be given to the action, but not to the referred activity.

5. Select the *:Desired Temperature* node and click  on its smart manipulator toolbar.

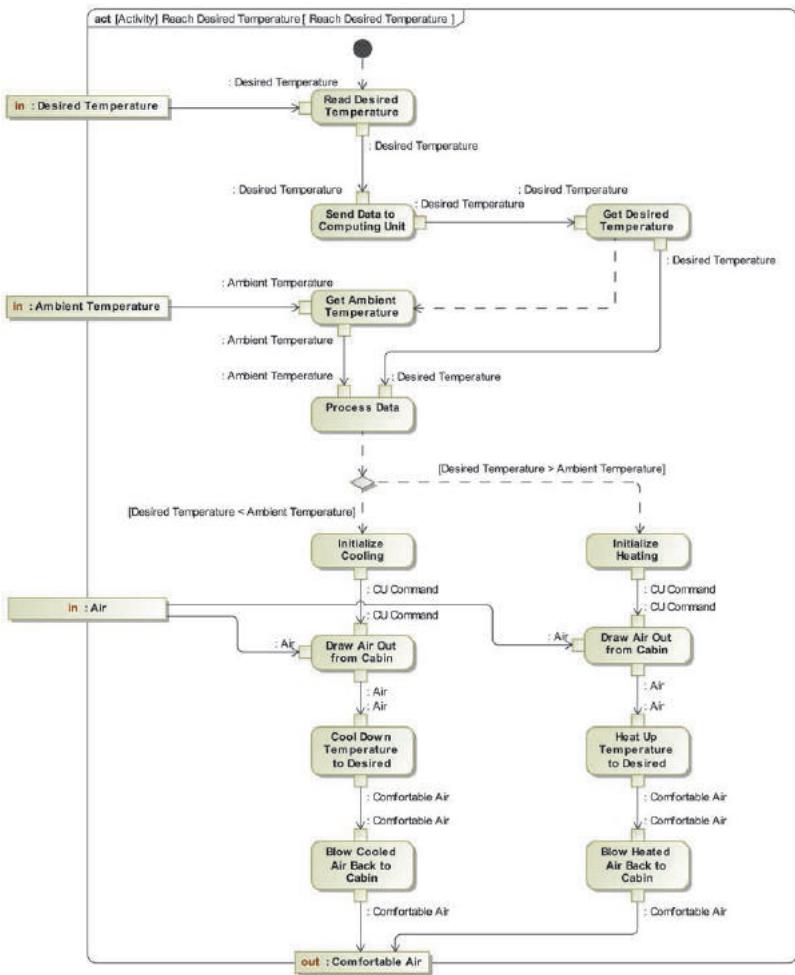
6. Click the *Read Desired Temperature* action. The object flow is created, as well as the input pin for the action. The type of that pin is automatically set by the type of the input parameter node.



i Remember that to hide the input pin name and show its type, you need to right-click that pin, click the **Show Name** check-box to clear the selection, and then click to select the **Show Type** check box.

7. Capture other items to build the complete scenario, which is displayed in the following figure using the following guidelines:

- To capture new actions, click the Control Flow or Object Flow on the smart manipulator toolbar. When you create the object flow, do not forget to right-click an empty place in the diagram pane and then select **Call Behavior Action** (otherwise the Central Buffer Node may appear on the diagram).
- To connect two actions, click the Control Flow or Object Flow on the smart manipulator toolbar of the source action, then click the target action.
- To set the type for a pin (unless it is not automatically set), drag the appropriate element (block or signal) from the Model Browser.
- To create the decision or merge node, right-click an empty place in the diagram pane and then select **Decision** or **Merge**.
- To specify a guard, select the relevant control flow, type the square bracket, and then specify the condition, such as *[Desired Temperature < Ambient Temperature]*. Keep in mind that you do not need to close the brackets manually. This is done automatically once you press Enter.
- To create an output pin and an object flow to the output parameter node, click the Object Flow on the smart manipulator toolbar of that action and then click the output parameter node you want to relate. The output pin is created in tandem and its type is set automatically by the type of the related output parameter node.
- 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 an empty place in the diagram pane and select that type.



Note that there are a few pins that are typed by the *CU Command* element, where *CU* stands for *Computing Unit*. This is actually a signal. It can be stored in the *3 Exchange Items* package, together with other signals and blocks that capture exchange items.

You can run the simulation to validate the specified behavior and see if your model is correct. To start the simulation session, click the Run button on the *Reach Desired Temperature* activity diagram toolbar. When in the Simulation panel, click the Start button to start the model execution.

This scenario enables us to presume that the Vehicle Climate Control Unit may consist of four subsystems: one for receiving data from outside the system, one for cooling the air, one for heating it, and one for controlling other systems. The next cell of the MagicGrid framework describes how to capture them.

Step 4. Creating an activity decomposition map

Now we will create an activity decomposition map for the *Provide Comfortable Temperature* activity and see which black- and white-box functions it was decomposed to. The activity decomposition map is one of the predefined relation maps available in the [modeling tool](#).

To create the activity decomposition map for the *Provide Comfortable Temperature* activity

1. Select the activity in the Model Browser:

- a. Press Ctrl + Alt + F.
- b. In the **Quick Find** dialog:
 - i. Make sure the **Type** option is selected.
 - ii. Type *Provide C*.
- c. When you see the *Provide Comfortable Temperature* activity in the search results list, select it and press Enter. The activity is selected in the Model Browser.

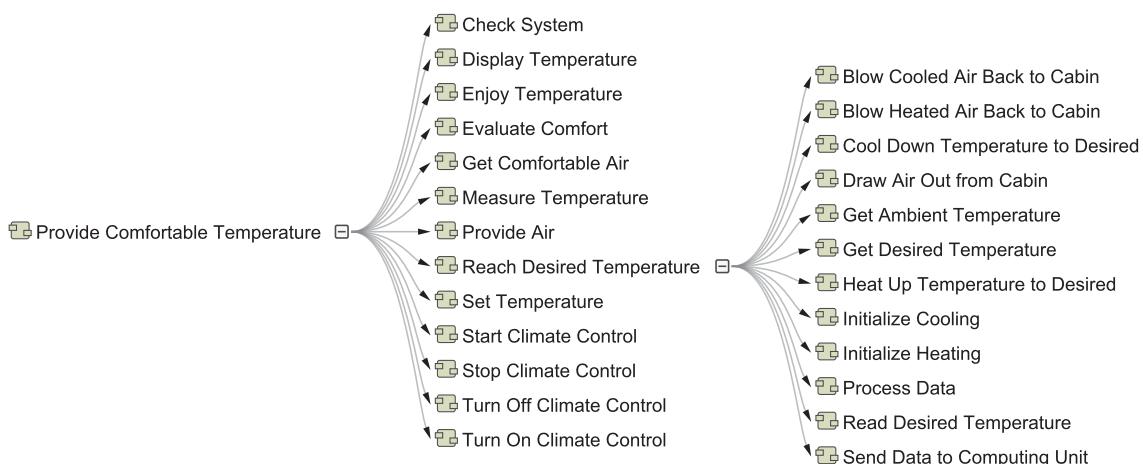
2. Right-click the activity and select **Create Diagram**.

3. In the search box, type *adm*, the acronym of the predefined Activity Decomposition Map, and then press Enter. The diagram is created.

(i) If there are no results in the diagram types list, click the **Expert** button below it. The list of available diagrams expands in the Expert mode.

4. Press Enter again.

(i) Note that the diagram is named after the enclosing activity. This name suits the diagram as well, so no changes are needed.



(i) If only the fragments of long element names are displayed, you can change the map settings to display the complete element names. For this, click the Options button on the relation map toolbar and click to unselect **Cut Element Names**.

Functional Analysis done. What's next?

White-box scenarios stimulate identification of conceptual subsystems of the **Sol**; thus, the next cell is **Conceptual Subsystems**.

Conceptual Subsystems

What is it?

While functional analysis (see Chapter [Functional Analysis](#)) helps identify conceptual subsystems (aka functional blocks), this cell captures them in the model. The conceptual subsystems should be considered as a group of interconnected and interactive parts that are responsible for performing one or more expected functions of the [SoI](#).

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

Once the conceptual subsystems are captured in the model, it is time to specify which system functions they perform. For this, functions are allocated to conceptual subsystems. Remember that the granularity of expected system behavior and structure must be consistent in each level of detail; thus, you cannot assign the function of the subsystem to its component or function of that component to the subsystem. Deeper analysis is only possible after you have completely finished the current level of granularity. The recommended correspondence between the granularity levels of behavioral and structural decomposition is displayed in Chapter [White-box perspective](#).

Inputs and outputs of the [SoI](#) are also considered in this cell. They are identified by analyzing the system context and use case scenarios, while others are determined considering the results of the functional analysis.

Overall, this cell produces:

- Inputs and outputs of the [SoI](#)
- Definitions of conceptual subsystems of the [SoI](#)
- Interactions:
 - Between conceptual subsystems and the outside of the [SoI](#)
 - Among conceptual subsystems
- Allocation of expected system functions to the conceptual structure of the [SoI](#)
- Structure decomposition map

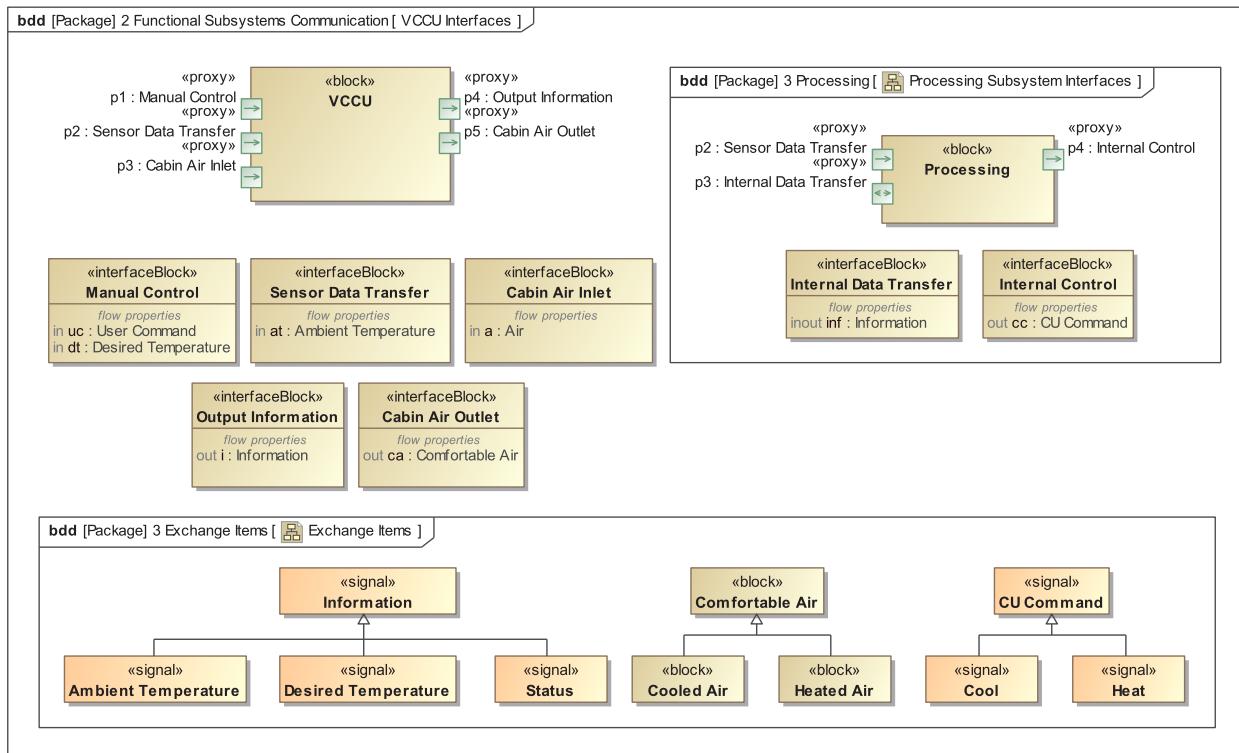
i It should be kept in mind that it is not always necessary to specify the conceptual architecture of the [SoI](#). For example, when MagicGrid is used to modernize an already existing system, the Conceptual Subsystems cell can be skipped to define the logical architecture of the SOI straightforwardly.

Who is responsible?

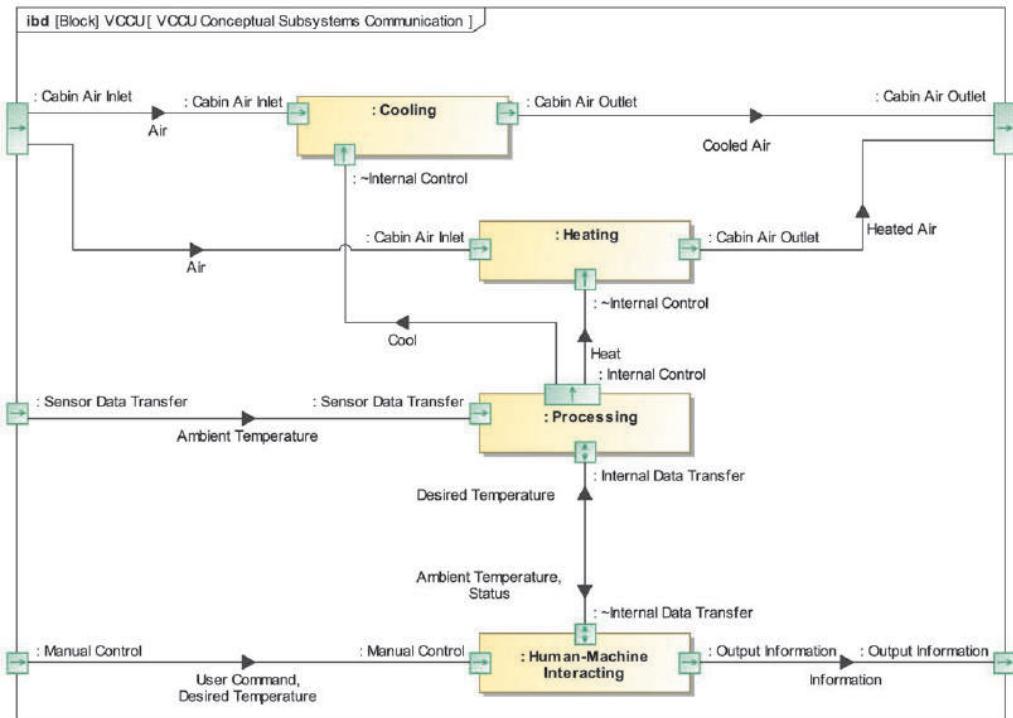
The structural decomposition of the system can be performed and its internal communications can be specified by the Systems Analyst or Systems Engineer.

How to model?

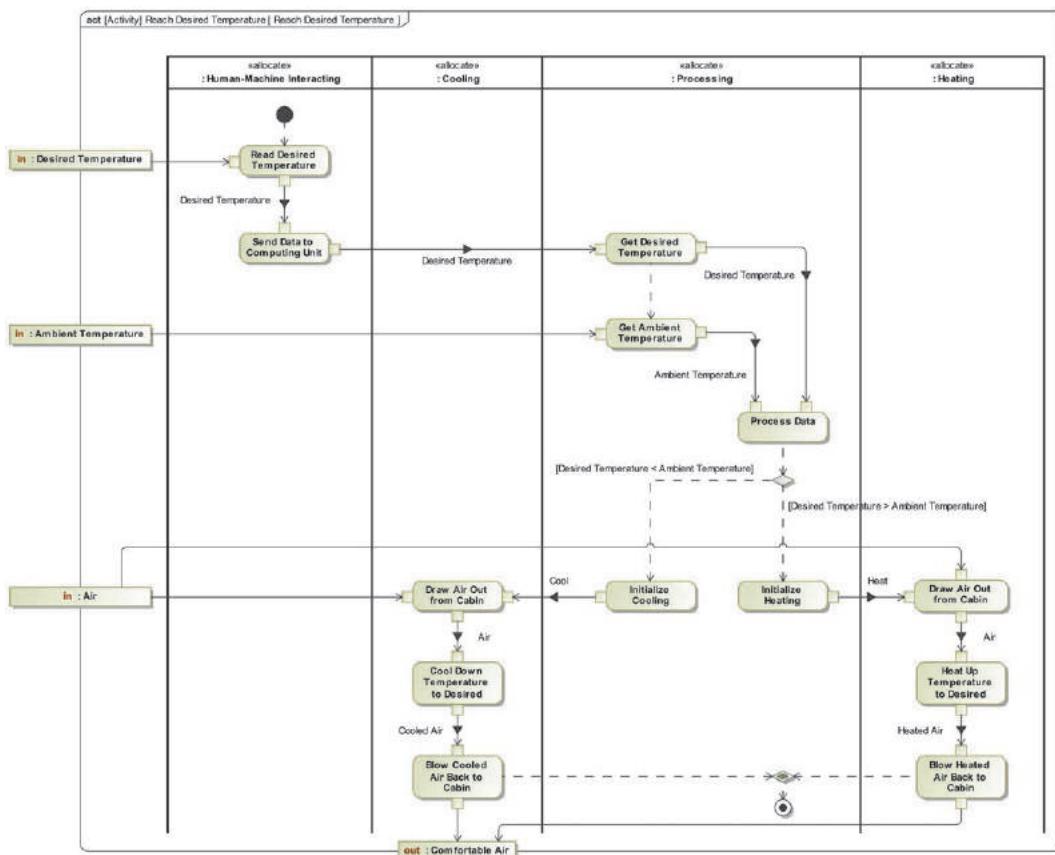
To capture inputs and outputs of the **Sol** or any of its conceptual subsystems, you can use the infrastructure of the **bdd**. These inputs and outputs can be specified as SysML flow properties. As defined by SysML, these can be grouped by interface blocks that are stored somewhere in the model or even in some external library. Proxy ports are used to relate interface blocks to the blocks that capture the structure of the **Sol**. The following figure shows the inputs and outputs of the Vehicle Climate Control Unit and one of its conceptual subsystems, the Processing subsystem.



To define the conceptual subsystems of the **Sol** and the interactions between them, you can utilize the infrastructure of the **ibd** created for the block that captures the **Sol** in your model. Conceptual subsystems, identified while performing the functional analysis (see Chapter [Functional Analysis](#)), can be specified as part properties of the block that represents the **Sol**. Connectors with one or more item flows between these part properties indicate interactions between appropriate conceptual subsystems. Connectors with item flows between part properties and the diagram frame represent interactions between conceptual subsystems and the outside of the **Sol**. Connections should be established via proxy ports. The following figure shows the conceptual subsystems of the Vehicle Climate Control Unit as well as their communication with the outside and in between.



After you have captured all the conceptual subsystems in the model, you can go back to the white-box scenario and allocate the white-box functions of the **Sol** to its conceptual subsystems in order to convey that each conceptual subsystem (aka functional block) is responsible for performing one or more white-box functions. Conceptual subsystems can be represented in the activity diagram as swimlanes. The following figure displays the subfunctions of the *Reach Desired Temperature* function, allocated to the conceptual subsystems of the **Sol**.

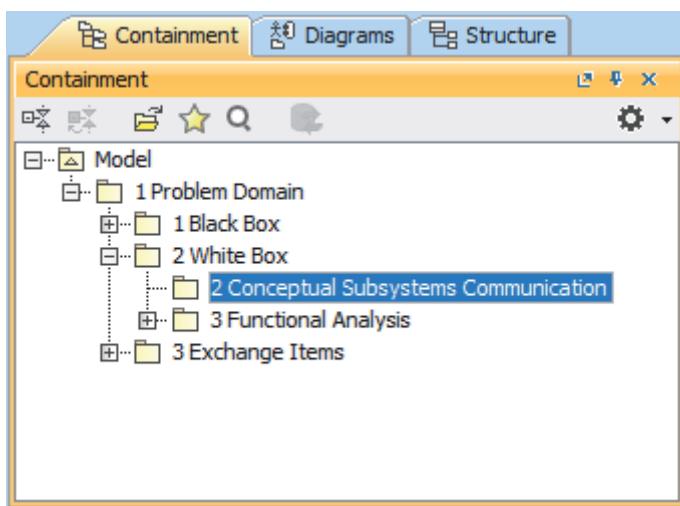


Tutorial

- Step 1. Organizing the model for conceptual subsystems communication
- Step 2. Creating a bdd for specifying conceptual interfaces
- Step 3. Capturing conceptual interfaces
- Step 4. Creating an ibd for capturing conceptual subsystems
- Step 5. Capturing conceptual subsystems
- Step 6. Specifying interactions between conceptual subsystems and the outside of the Sol
- Step 7. Specifying interactions among conceptual subsystems
- Step 8. Allocating functions by conceptual subsystems and synchronizing item flows

Step 1. Organizing the model for conceptual subsystems communication

Following the structure of the MagicGrid framework, model artifacts that capture conceptual subsystems communication should be stored in a separate package inside the *2 White Box* package. We recommend naming the package *2 Conceptual Subsystems Communication*.



To organize the model for the conceptual subsystems communication

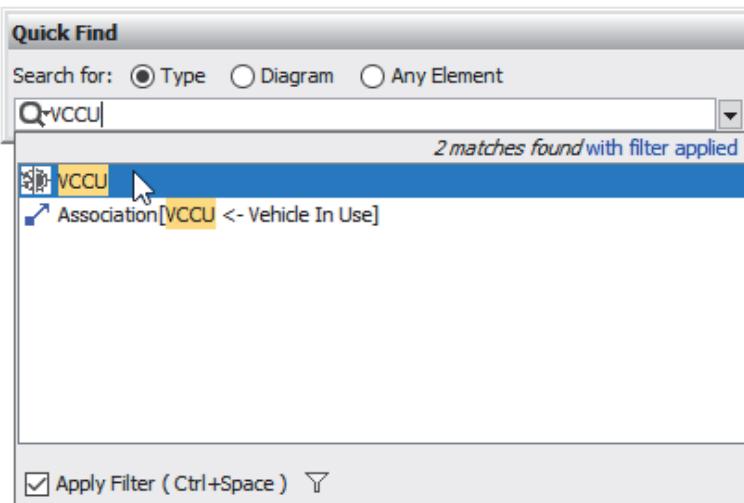
1. Right-click the *2 White Box* package and select **Create Element**.
2. In the search box, type *pa* (the first two letters of the element type *Package*) and press Enter.
3. Type *2 Conceptual Subsystems Communication* to specify the name of the new package and press Enter.

Step 2. Creating a bdd for specifying conceptual interfaces

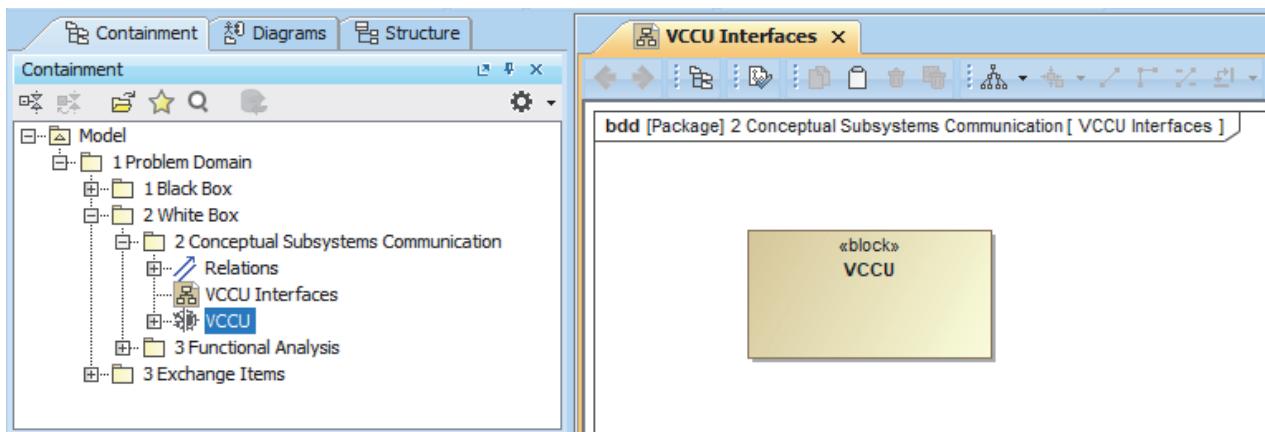
To begin specifying conceptual interfaces, which capture inputs and outputs of the *Sol*, you first need to create a bdd and display the block that represents the *Sol* on the diagram pane. Before this, however, you should change the location of that block in your model by moving it to the package *2 Conceptual Subsystems Communication*. Since we've started the white-box analysis, it is no longer a black box.

To create and prepare a **bdd** for specifying **Sol** interfaces

1. Create the bdd:
 - a. Right-click the *2 Conceptual Subsystems Communication* 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 *VCCU Interfaces* to specify the name of the new diagram and press Enter again.
2. In the Model Browser, select the *VCCU* block using the quick find capability:
 - a. Press Ctrl + Alt + F. The **Quick Find** dialog opens.
 - b. Type *VC*.
 - c. When you see the *VCCU* block selected in the search results list (see the following figure), press Enter. The *VCCU* block is selected in the Model Browser.



3. Drag the *VCCU* block to the newly created package *2 Conceptual Subsystems Communication* (see **step 1 of this cell tutorial**). If a message appears, asking if you want to move block's relationships together, click **Yes**.
4. Drag the *VCCU* block to the newly created diagram pane. The shape of the block appears on the diagram.



Step 3. Capturing conceptual interfaces

By analyzing the system context (see Chapter [System Context](#)), we can conclude the following:

- The Vehicle Climate Control Unit is controlled by the Vehicle Occupant (driver or passenger), who can Turn On and Turn Off the system, as well as specify the Desired Temperature.
- The Vehicle Climate Control Unit takes the Air from the Cabin and provides back the Comfortable Air.
- The Temperature Sensor of the Vehicle provides the Ambient Temperature of the Cabin Vehicle Climate Control Unit.
- The Vehicle Climate Control Unit displays the Information (for example, status information, temperature in the cabin).

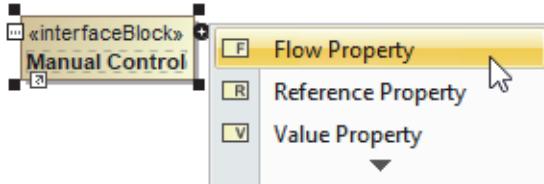
This information serves as input to specify conceptual interfaces of the [Sol](#), for defining its communication with the outside: supposed users and other entities of the *Vehicle In Use* system context. Conceptual interfaces for internal communication will be considered and captured later, after we define conceptual subsystems (also known as functional blocks) of the [Sol](#).

Remember that inputs and outputs can be captured as flow properties. Flow properties with *in* direction indicate inputs, and flow properties with *out* direction indicate outputs. Flow property names are not as relevant as their types. Blocks and signals that capture exchange items of the *Vehicle In Use* system context become flow property types.

A single flow property always belongs to some interface block. A proxy port is a usage of that interface block on the block, which captures the [Sol](#). Proxy port names are relevant, if more than one proxy port is typed by the same interface block; otherwise, their names are not important.

To create the interface block for providing manual control and relate it to the *VCCU* block

1. Open the *VCCU Interfaces* diagram, if not opened yet.
2. Click the **Interface Block** button on the diagram palette and then click an empty place on the diagram pane. An unnamed interface block is created in the model, directly under the *2 Conceptual Subsystems Communication* package, and displayed on the diagram.
3. Type *Manual Control* to name the interface block and press Enter.
4. Click the Create Property button  on the interface block shape and select **Flow Property**.



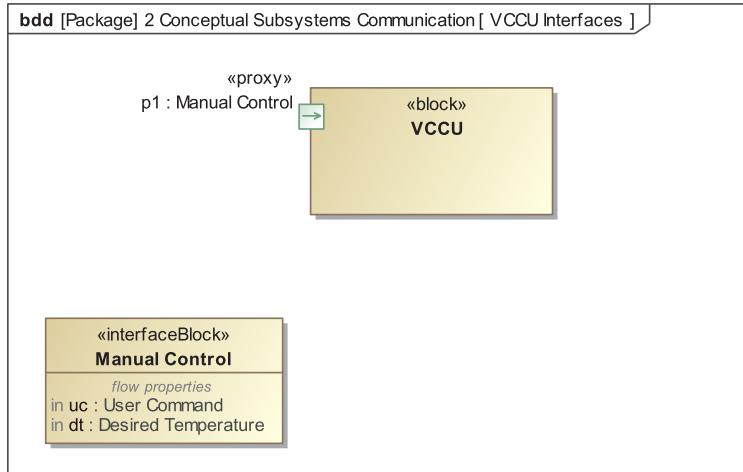
5. Directly on the shape of the interface block, do the following:
 - a. Remove the *out* part of the flow property direction indicator to specify that the direction of the flow property is *in*.
 - b. Type *uc* to specify the flow property name.
 - c. Type *:User* and pres Ctrl + Spacebar to open the list of possible flow property types. Once you see the *User Command* signal in that list, select it. The type is set.



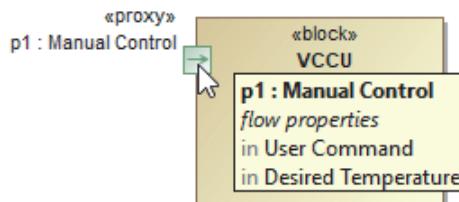
6. Repeat steps 4 and 5 to create another flow property you see in the following figure.



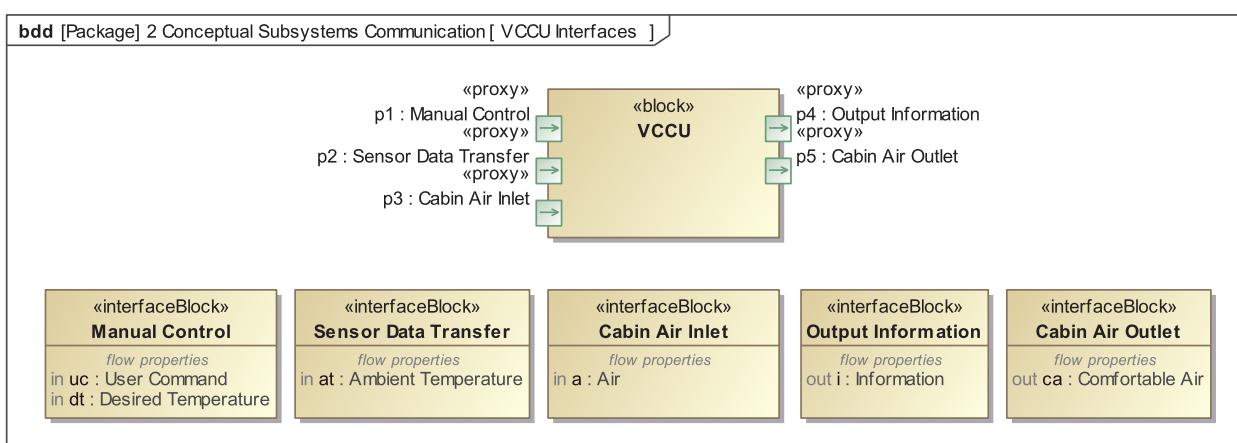
7. Select the *Manual Control* interface block on the diagram and drag it onto the *VCCU* block. As a result, a new proxy port, typed by the *Manual Control* interface block, is created for the *VCCU* block. As can see, it is decorated with an arrow indicating that this port is for inputs to the system of interest.



- i** If you hover over the proxy port shape, the tooltip shows which inputs can be provided to the system over this proxy port.



Following the procedure above, create the interface blocks with flow properties (shown in the following figure) and then relate them to the *VCCU* block by creating proxy ports.

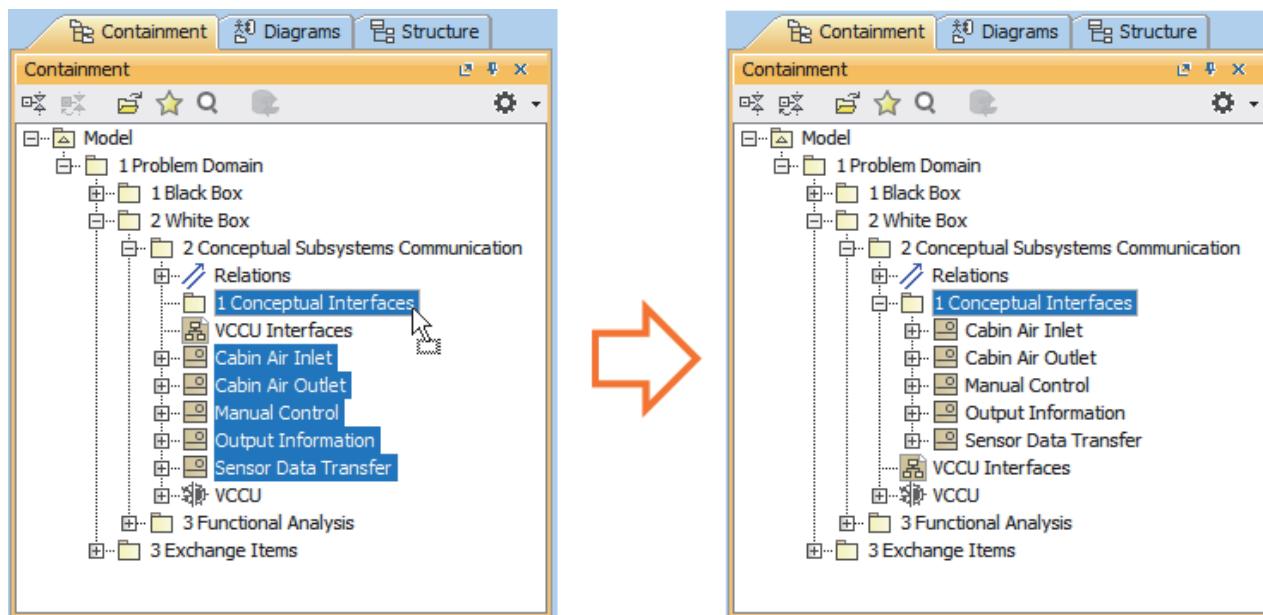


i Once finished, the *VCCU Interfaces* diagram can be marked as complete. A complete diagram excludes unwanted layout changes that may occur as your model evolves. Model updates that concern the complete diagram (for example, new properties of the block that resize its shape and make it overlap with other shapes) are suspended there until you allow them to be displayed. To mark the *VCCU Interfaces* diagram as complete, right-click anywhere on the diagram pane and then select **Complete**. The notification indicating that the diagram is suspended to display changes in the model appears at the top of it. For more information about this feature, refer to the latest documentation of the [modeling tool](#).

The following figure displays the Blackbox ICD Table, an alternative view of the Vehicle 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](#).

#	Port Name	Port Type	Type Features	Direction
1	p1	Manual Control	[F] in dt : Desired Temperature [F] in uc : User Command	in
2	p2	Sensor Data Transfer	[F] in at : Ambient Temperature	in
3	p3	Cabin Air Inlet	[F] in a : Air	in
4	p4	Output Information	[F] out i : Information	out
5	p5	Cabin Air Outlet	[F] out ca : Comfortable Air	out

To keep the model well-structured, we recommend storing all the interface blocks in a separate package. For this, you need to create the *1 Conceptual Interfaces* package directly under the *2 Conceptual Subsystems Communication* package and then drag all the interface blocks to it.



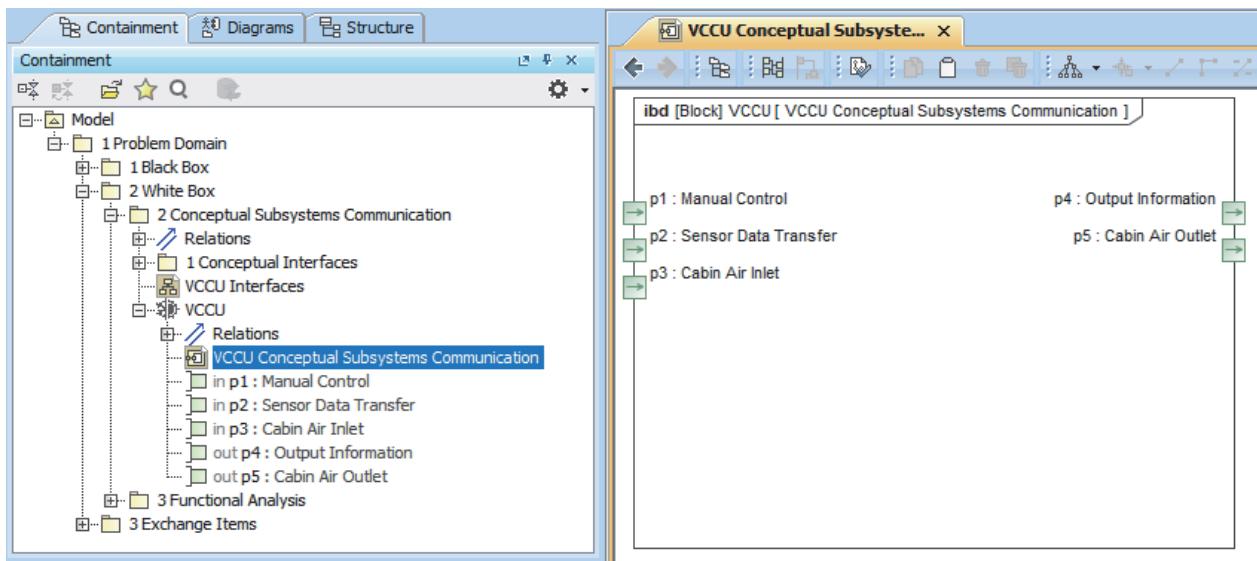
Step 4. Creating an ibd for capturing conceptual subsystems

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

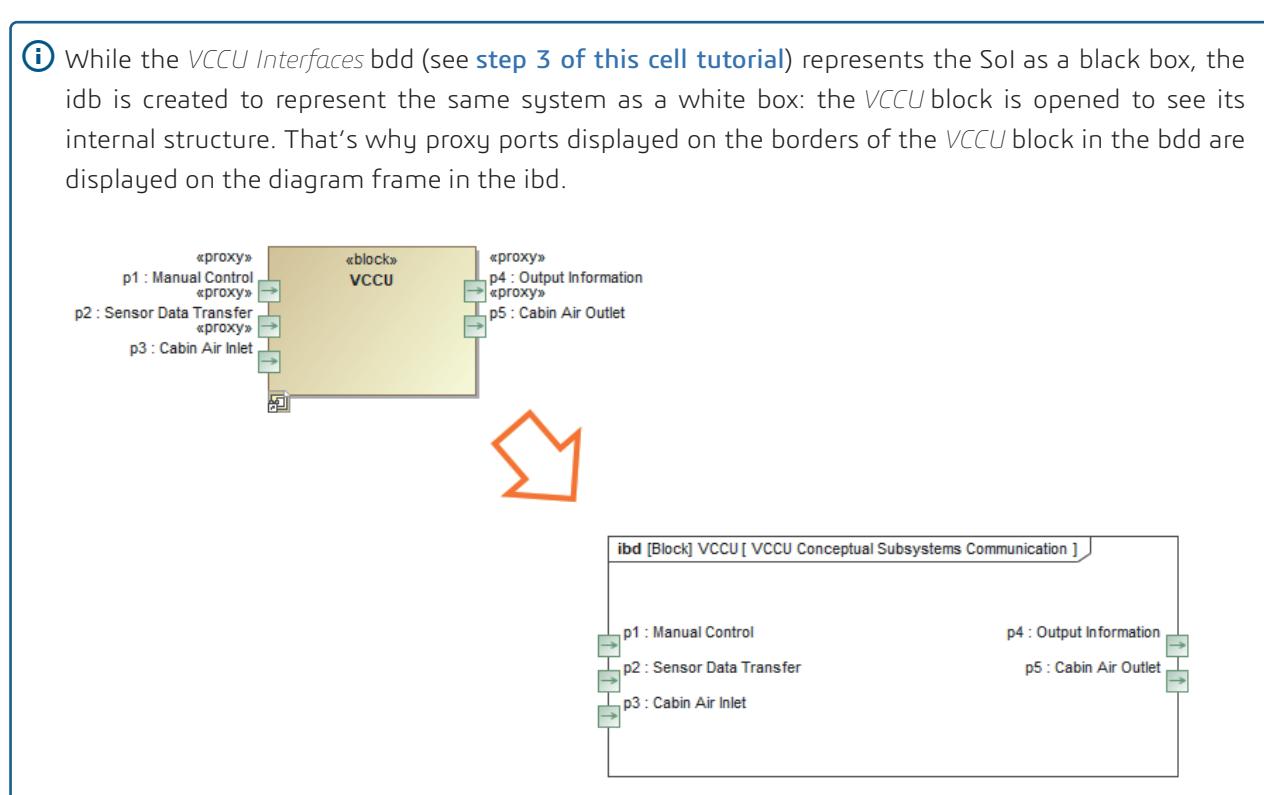
To create an ibd for capturing conceptual subsystems of the Sol

1. On the *VCCU Interfaces* diagram, select the shape of the *VCCU* block and click the SysML Internal Block Diagram button  on its smart manipulator. A new ibd named after the enclosing block is created, and the **Display Parts/Ports** dialog opens.
2. Make sure all the proxy ports of the *VCCU* block are selected and click **OK**. The proxy ports are displayed on the newly created diagram frame.
3. Type *VCCU Conceptual Subsystems Communication* to specify the name of the new diagram and press **Enter** again.

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



- i** While the *VCCU Interfaces* bdd (see [step 3 of this cell tutorial](#)) represents the Sol as a black box, the ibd is created to represent the same system as a white box: the *VCCU* block is opened to see its internal structure. That's why proxy ports displayed on the borders of the *VCCU* block in the bdd are displayed on the diagram frame in the ibd.



Step 5. Capturing conceptual subsystems

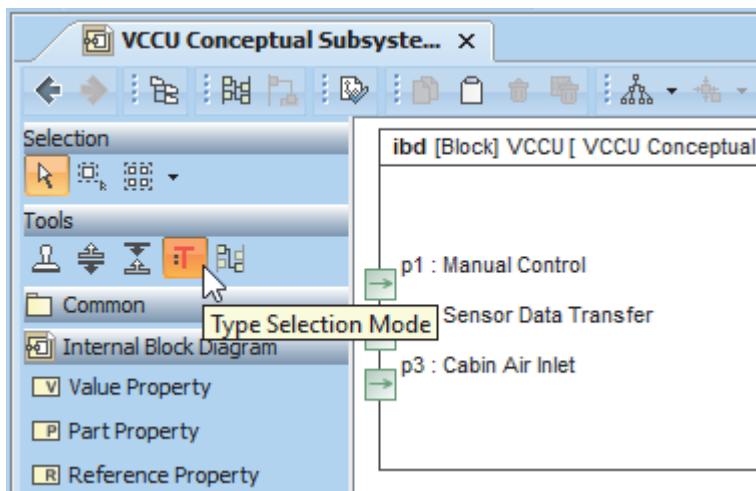
As the functional analysis has revealed (see Chapter [Functional Analysis](#)) the Vehicle Climate Control Unit consists of these conceptual subsystems:

- Cooling
- Heating
- Processing
- Human-Machine Interacting

They can be specified as part properties of the *VCCU* block. Part properties don't need names, but they should be typed by blocks that capture the conceptual subsystems listed above.

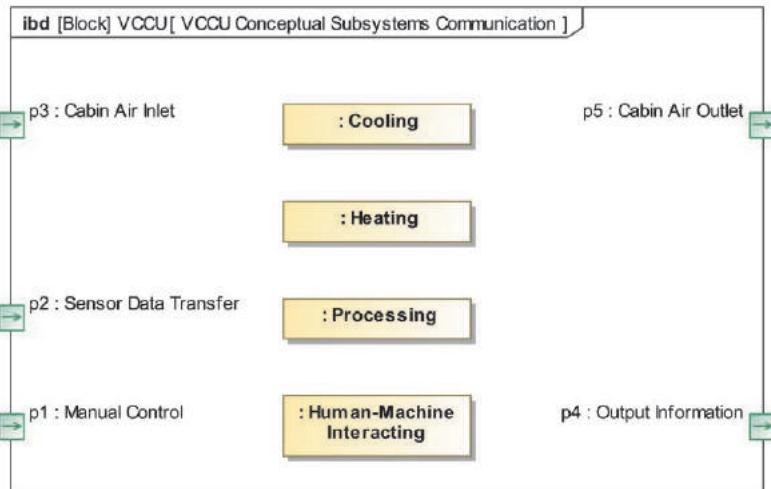
To specify the conceptual subsystems

1. Open the *VCCU Conceptual Subsystems Communication* ibd, created in [step 4 of this cell 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.

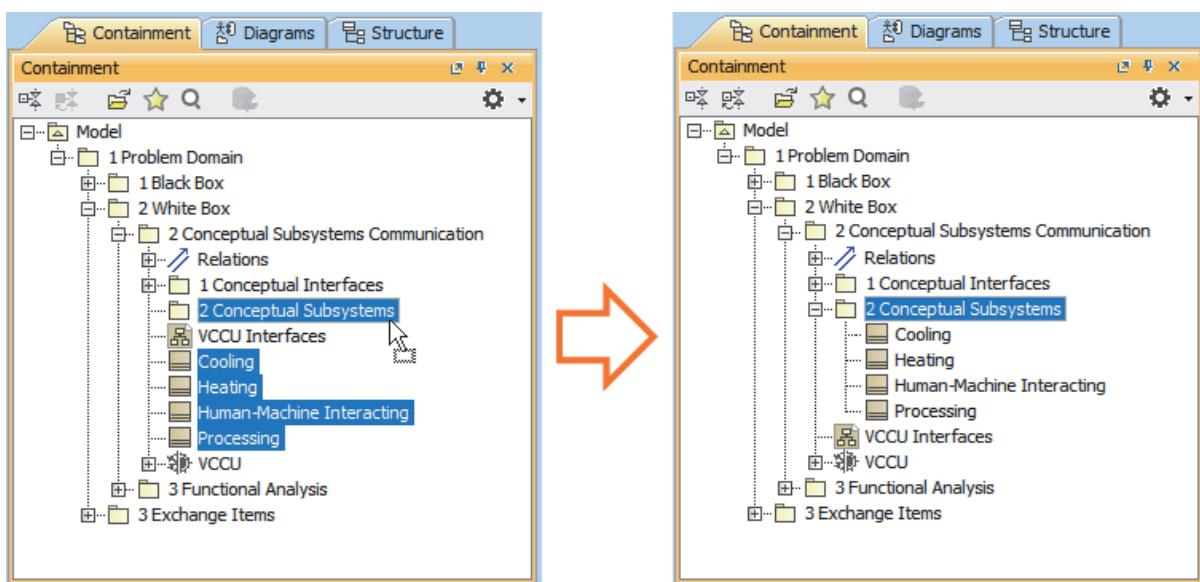


3. Click the **Part Property** button on the diagram palette and then click an empty place on the diagram pane. An unnamed part property is created, and the list of existing blocks to type it is offered.
4. Type *Cooling* next to the colon (":") directly on the part property shape and press Enter. The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

When you're done, your *VCCU Conceptual Subsystems Communication* ibd should look very similar to the one in the following figure.



To keep the model well-structured, we recommend storing all the blocks that capture conceptual subsystems in a separate package. For this, you need to create the *2 Conceptual Subsystems* package directly under the *2 Conceptual Subsystems Communication* package and then drag all the blocks to it.



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

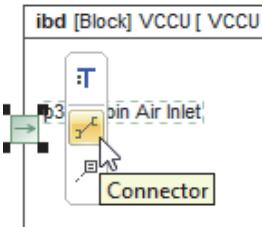
Step 6. Specifying interactions between conceptual subsystems and the outside of the Sol

In this step, we focus on specifying interactions between conceptual subsystems (aka functional blocks) of the Vehicle Climate Control Unit and outside it. An interaction between the subsystem and the outside of the **Sol** can be specified as a connector with one or more item flows, linking the appropriate part property and the diagram frame (remember, the diagram frame in the ibd represents the borders of the **Sol**). In this cell, unlike the System Context cell (see Chapter [System Context](#)), connectors are established over compatible proxy ports. Information, matter, or energy exchanged over these connectors can be specified as items conveyed by item flows.

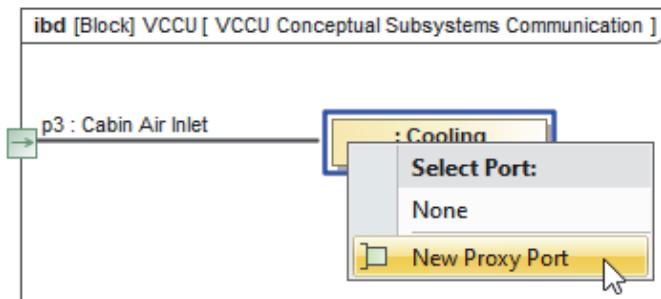
Let's start with specifying that the Cooling subsystem takes the Air from the outside of the **Sol** and provides Comfortable Air back.

To specify the interactions between the Cooling subsystem and the outside of the **Sol**

1. Select the proxy port typed by the *Cabin Air Inlet* interface block and click Connector button  on its smart manipulator toolbar.

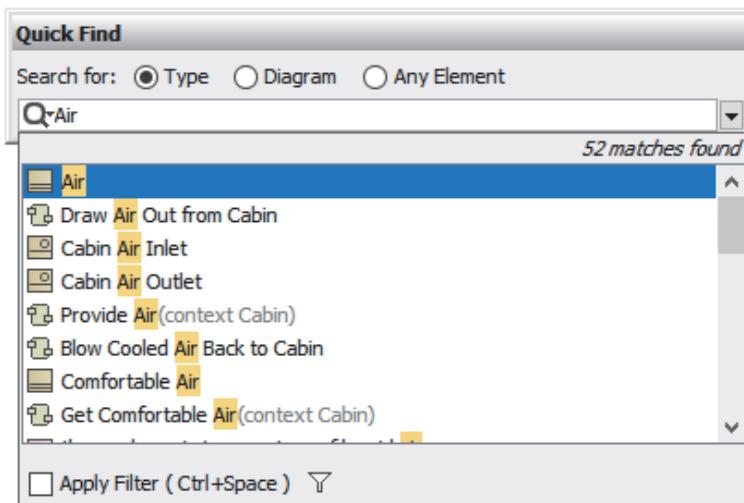


2. Click the *:Cooling* part property and select **New Proxy Port** (see the following figure). A new compatible proxy port is created. As shown below, it is typed by the same interface block.



3. In the Model Browser, select the *Air* block. For this, use the quick find capability:

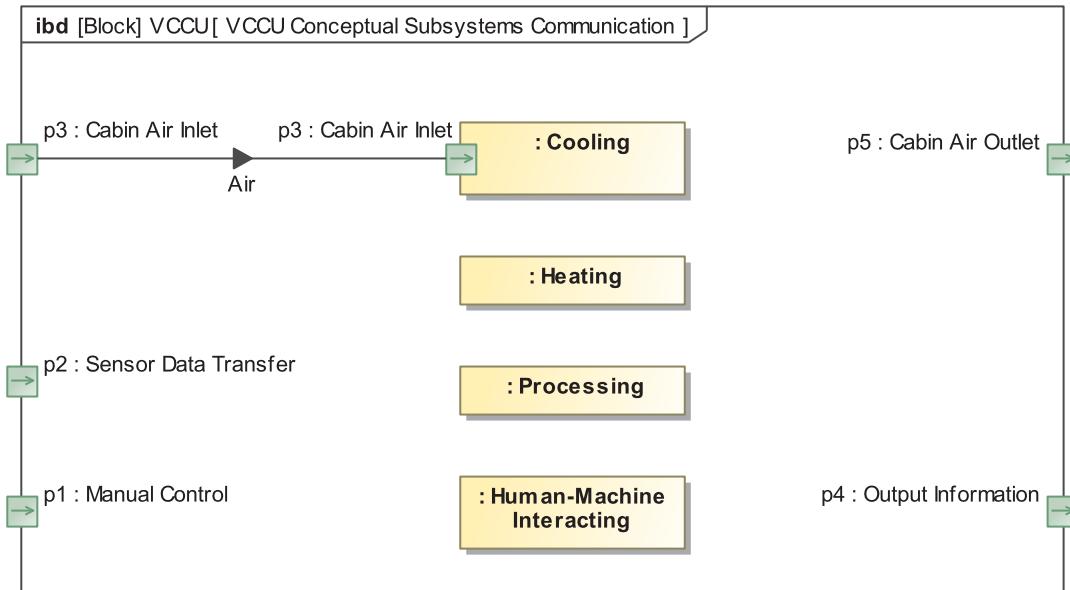
- a. Press Ctrl + Alt + F. The **Quick Find** dialog opens.
- b. Type *Air*.
- c. When you see the *Air* block selected in the search results list (see the following figure), press Enter. The *Air* block is selected in the Model Browser.



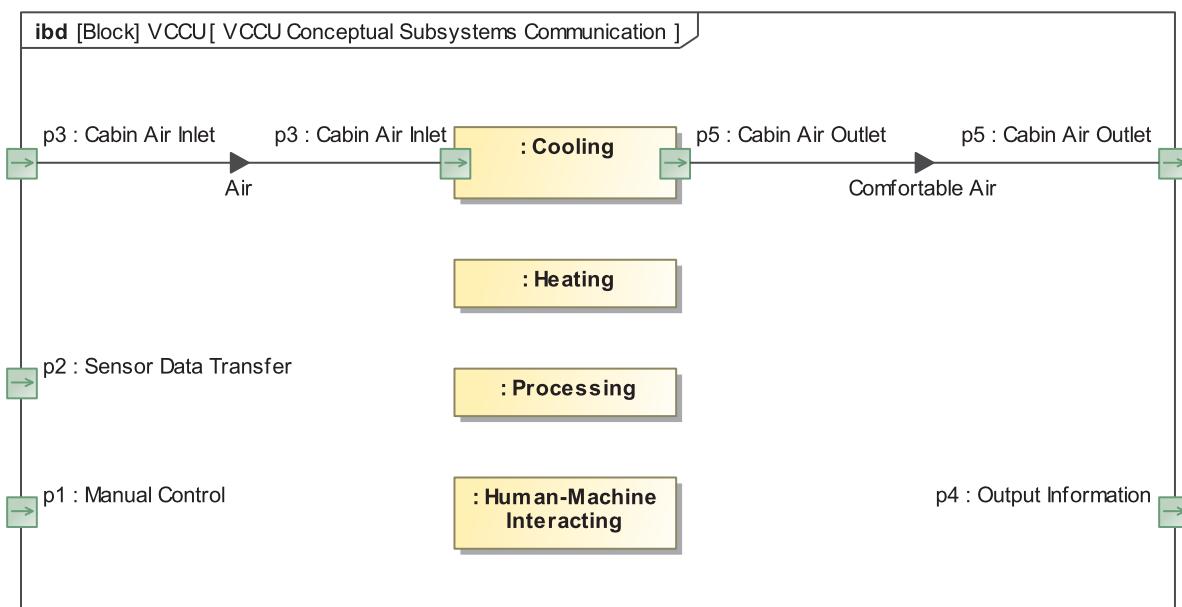
4. Drag the *Air* block to the newly created connector on the open diagram pane.

i As defined by SysML, the element you want to specify as the item flowing over the connector must be the type (or sub-type) of the flow property that determines the input of the part property on one connector end and the output of the part property on another, and vice versa. In this case, the *Air* block is the flow property type and can be set as the item flowing over the relevant connector.

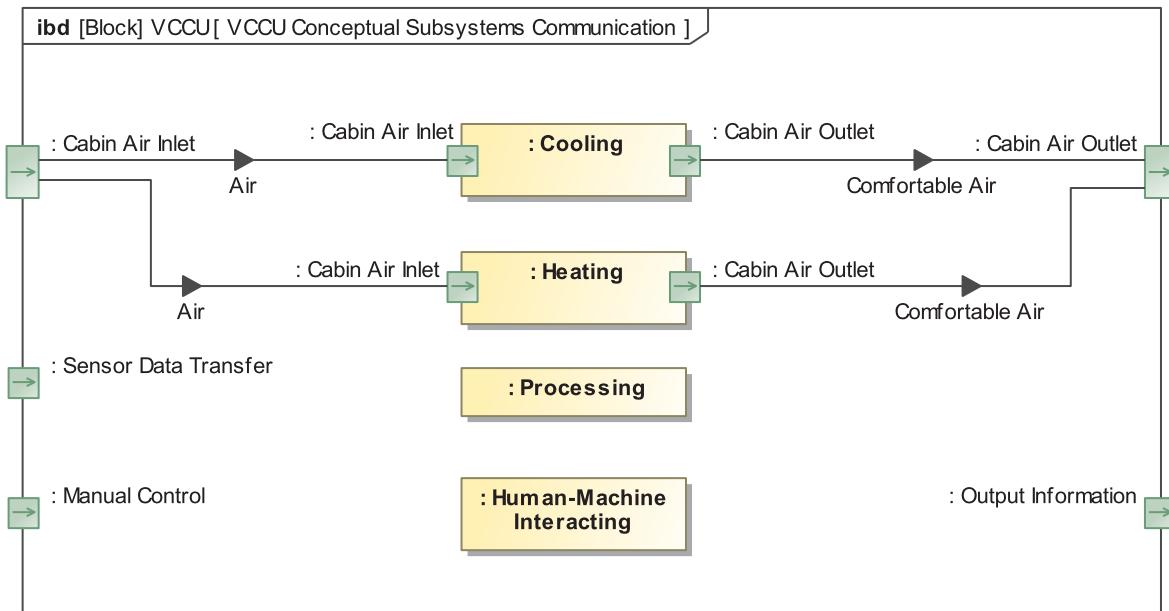
5. Don't make any changes in the open dialog; click **Finish**. The *Air* block is specified as the item flowing over the newly created connector.



6. Repeat steps 1 and 2 to create a connector between the proxy port typed by the *Cabin Air Outlet* interface block and the *:Cooling* part property.
7. Repeat step 3 to select the *Comfortable Air* block in the Model Browser.
8. Drag the block to the newly created connector.
9. In the open dialog, change the direction of the flow to **From Cooling System To VCCU** and click **Finish**. The *Comfortable Air* block is specified as the item flowing over the newly created connector.



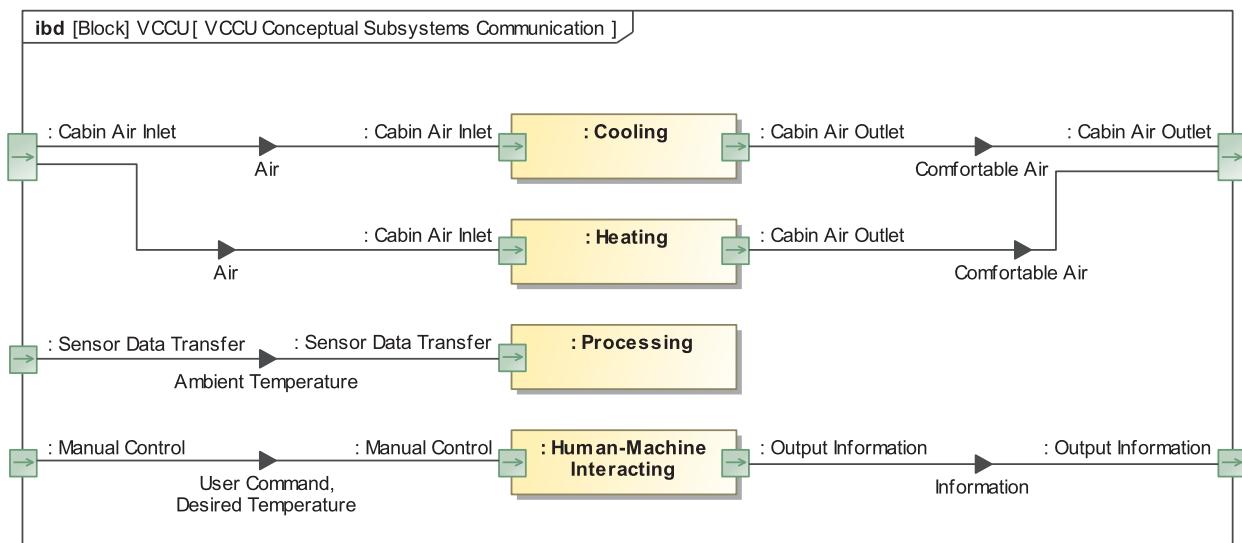
Interactions between the Heating subsystem and the outside are very similar. They can be specified in an appropriate way. Once you're done, your diagram should look like this.



- i** As you can see, the proxy port names are hidden in this diagram. This greatly increases the readability of this diagram. To hide the names of proxy ports, do the following:
1. Press Alt and hold it down while you select any proxy port; all proxy ports in that diagram are selected.
 2. Right-click the selection and then click to clear the **Show Name** check box.

Now specify that the Processing subsystem takes the Ambient Temperature from the outside, and the Handling Human-Machine Interactions subsystem accepts the User Commands and Desired Temperature from there. The latter provides various Information to the outside, too.

After you specify this by following the instructions above, your *VCCU Conceptual Subsystems Communication* diagram should look very similar to the one below.



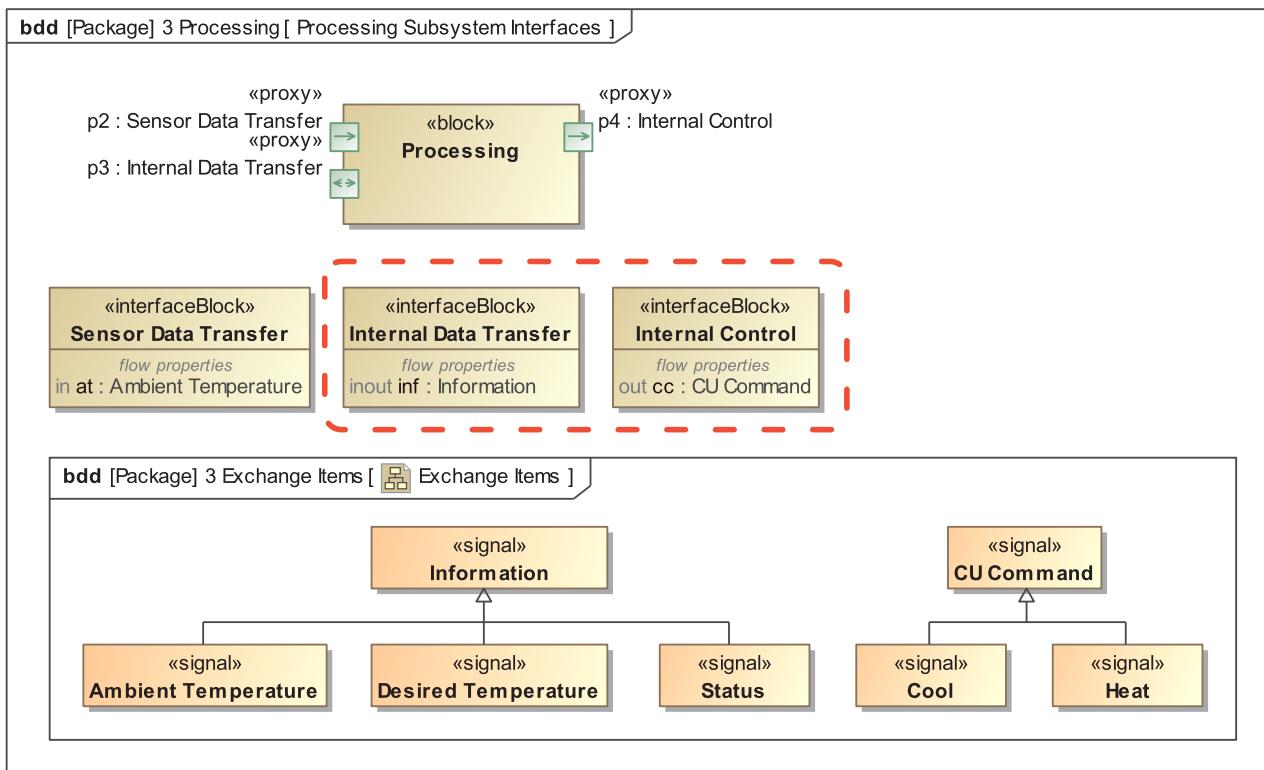
Step 7. Specifying interactions among conceptual subsystems

In this step, we will specify interactions among conceptual subsystems of the **Sol**. As in the previous step, these interactions can be specified as connectors established between appropriate part properties over compatible proxy ports. Information, matter, or energy exchanged over these connectors can be specified as items conveyed by item flows.

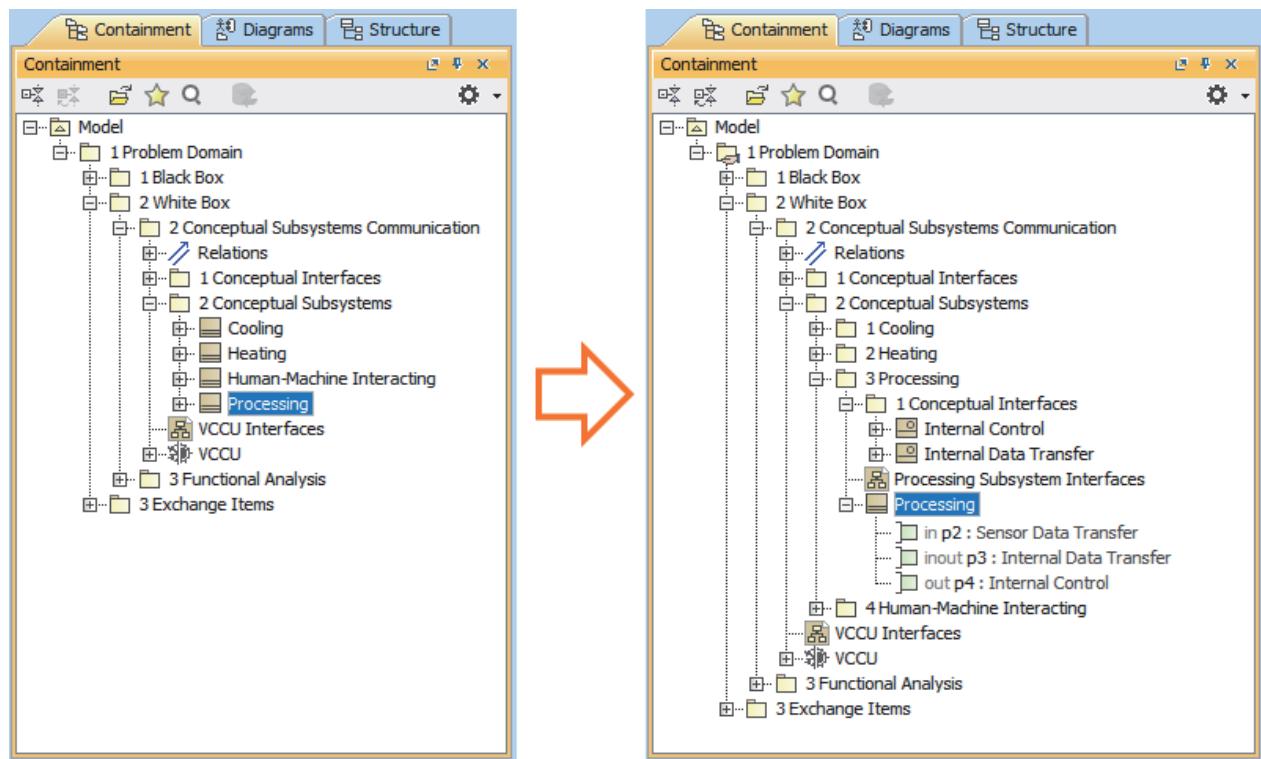
The functional analysis enables us to identify the following interactions:

- The Processing subsystem takes the Desired Temperature from the Human-Machine Interacting subsystem.
- If the Processing subsystem identifies that the Desired Temperature is lower than the Ambient Temperature, it initiates the Cooling subsystem.
- If the Processing subsystem identifies that the Desired Temperature is higher than the Ambient Temperature, it initiates the Heating subsystem.
- The Human-Machine Interacting subsystem takes various Information produced by the Processing subsystem to provide it to the outside.

The list above requires you to capture a few more interface blocks. You can see them highlighted in the following figure, which displays the *Processing Subsystem Interfaces* diagram.



We assume that you create this diagram and rearrange the model as it is displayed in the following figure, on your own.



As you can see, the *CU Command* signal has two sub-types: the *Cool* and *Heat* signal. With this hierarchy of signals, you can specify exactly which signal is flowing through the same conceptual interface in each particular case. It is the *Cool* signal between the *:Processing* and the *:Cooling* part properties, and the *Heat* signal between the *:Processing* and the *:Heating* part properties.

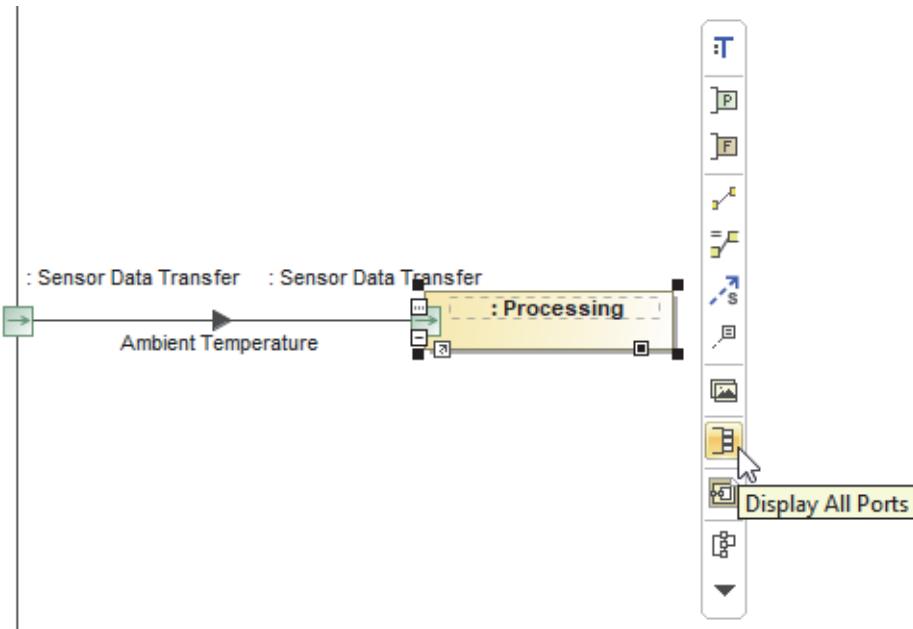
The *Information* signal has sub-types, too. These are the *Ambient Temperature*, *Desired Temperature*, and *Status* signals. We need them if we want to be more specific in specifying which kind of information can go through the proxy ports typed by the *Output Information* or *Internal Data Transfer* interface blocks.

Sub-types can be specified in a separate bdd, the *Exchange Items* diagram. You already know how to create a new bdd, display existing and create new elements there, and establish generalization relationships between them. Therefore, we assume that you do this on your own. To display the contents of the *Exchange Items* diagram within the *Processing Subsystem Interfaces* diagram, you need to drag the *Exchange Items* diagram from the Model Browser onto the pane of the *Processing Subsystem Interfaces* diagram. Then, on the smart manipulator toolbar of the *Exchange Items* diagram shape, click the Show Diagram Overview Content button .

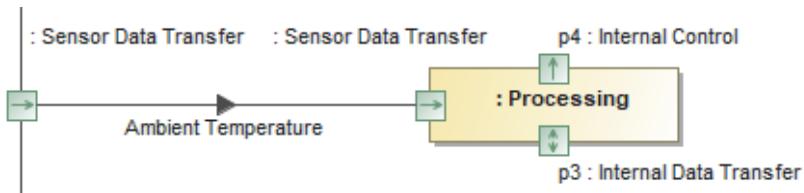
Now let's move on and specify the interactions among conceptual subsystems of the Vehicle Climate Control Unit.

To specify interactions between the Handling Human-Machine Interactions and the Processing subsystems

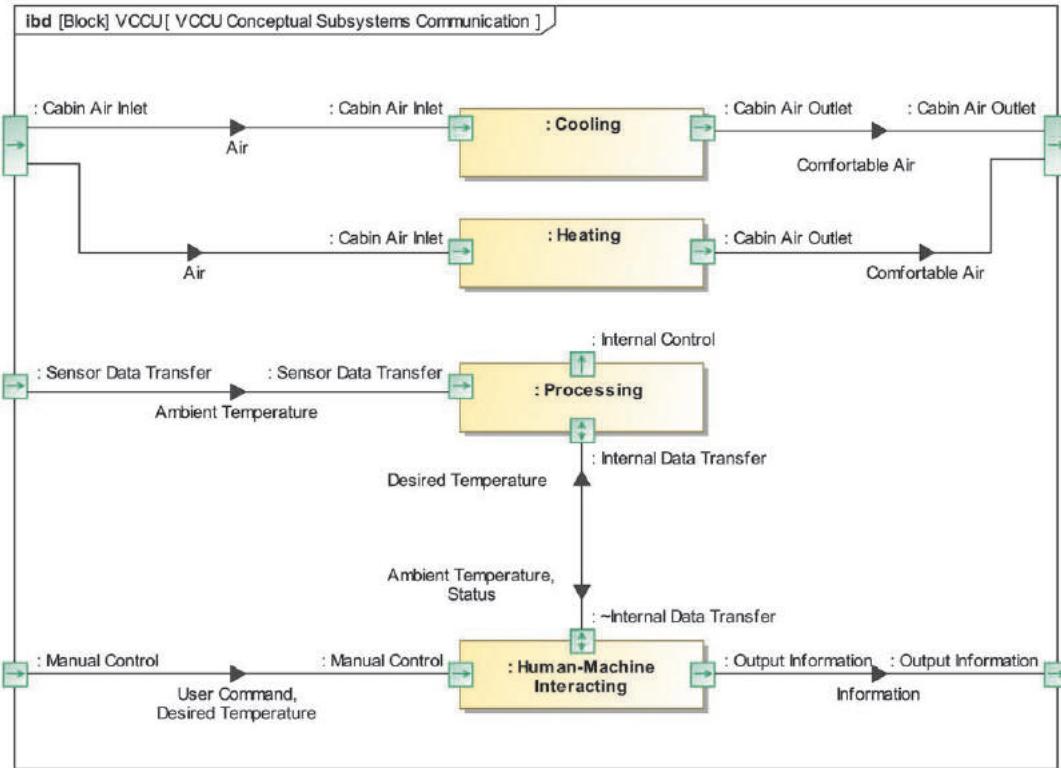
1. Open the *VCCU Conceptual Subsystems Communication* ibd, if not yet opened.
2. Select the *:Processing* part property and click Display All Ports button  on its smart manipulator toolbar.



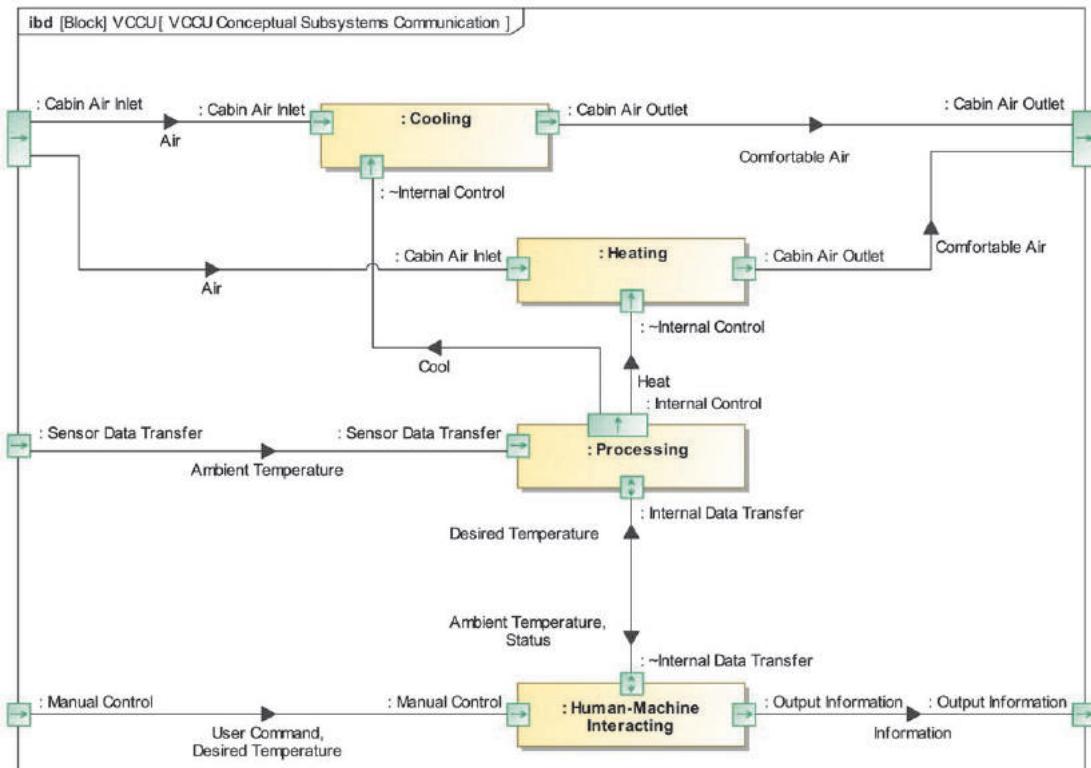
Two more proxy ports are displayed.



3. Select the proxy port typed by the *Internal Data Transfer* interface block and click the Connector button , its smart manipulator toolbar.
4. Click the *:Human-Machine Interacting* part property and select **New Proxy Port**. A new compatible proxy port is created. As you can see, it is typed by the same interface block.
5. In the Model Browser, under the *3 Exchange Items* package, select the *Desired Temperature* signal and drag it to the newly created connector.
6. In the open dialog, change the direction of the flow to **From Human-Machine Interacting to Processing** and click **Finish**. The *Desired Temperature* signal is specified as the item flowing over the newly created connector.
7. In the Model Browser, under the *3 Exchange Items* package, select the *Ambient Temperature* signal and drag it to the newly created connector.
8. Don't change anything in the open dialog and click **Finish**. The *Ambient Temperature* signal is specified as the item flowing over the newly created connector.
9. Repeat steps 7 and 8, to specify the *Status* signal as the item flowing over the newly created connector.

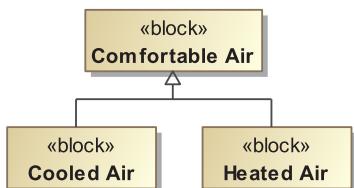


In the same manner, specify the interactions between the Processing subsystem and the Heating subsystem as well as between the former and the Cooling subsystem. Once you're done, your diagram should look very similar to the one below.



To be more precise, you can also specify that the Comfortable Air flowing out of the Cooling subsystem is Cooled Air, and the one flowing out of the Heating subsystem is Heated Air. Appropriate blocks should be

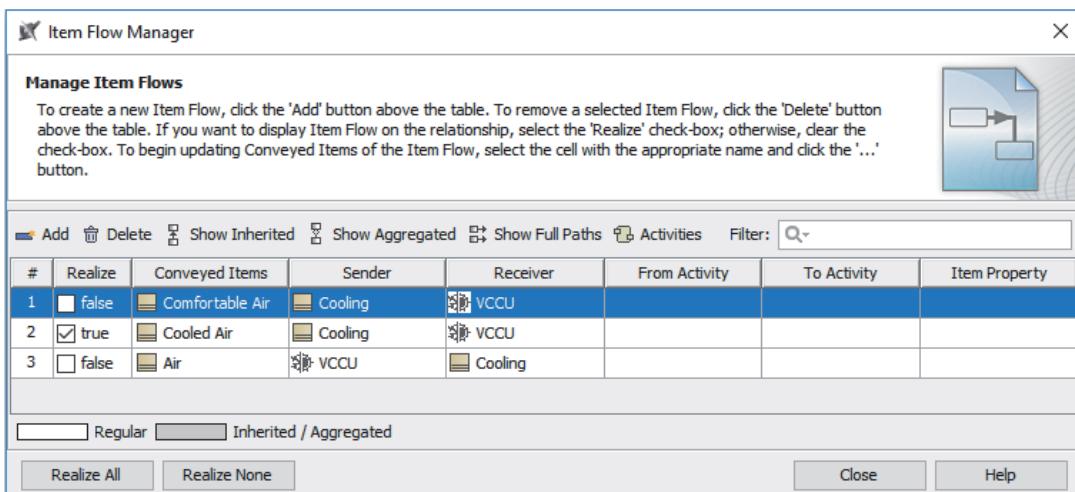
created as sub-types of the *Comfortable Air* block in the *Exchange Items* diagram: the *Cooled Air* block and the *Heated Air* block.



Since you have two sub-types of the *Comfortable Air* block, you can update the items flowing over the connectors between the *:Cooling* part property and the diagram frame as well as between the *:Heating* part property and the diagram frame. You already know how to do this. As you no longer need the *Comfortable Air* block flowing over the connector, you can simply get rid of it.

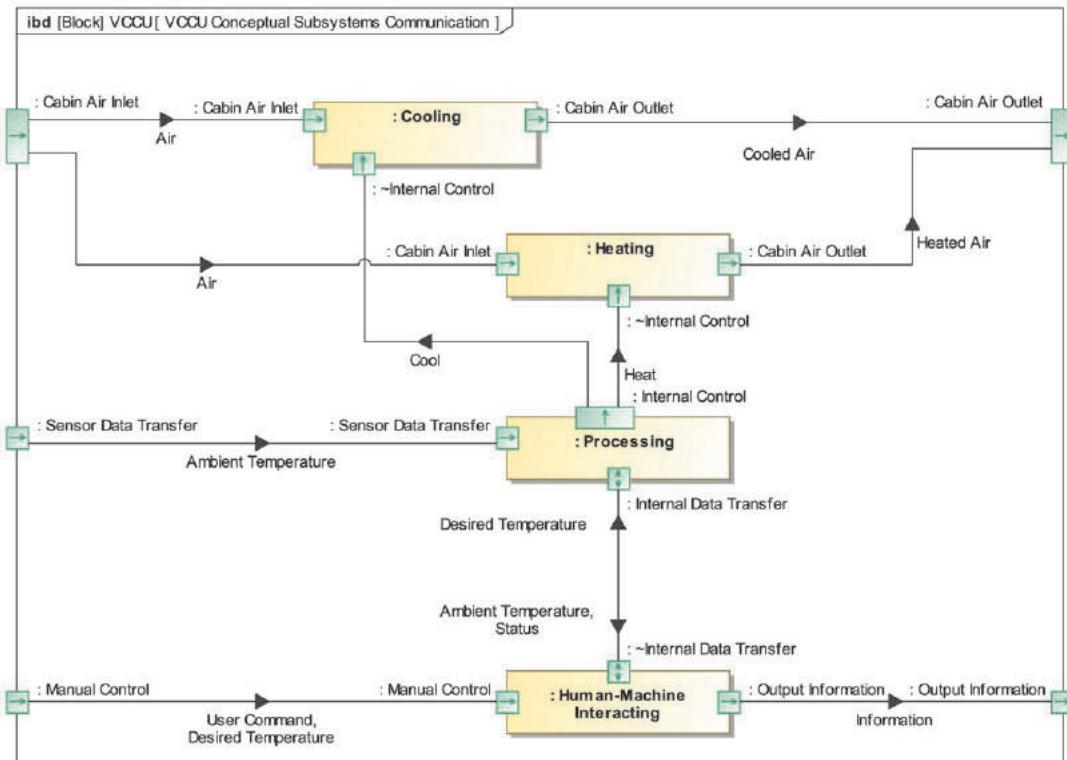
To remove the item flow

1. Select the relevant connector and click the Item Flow Manager button  on its smart manipulator toolbar.
2. In the open dialog, click to clear the selection of the row where the conveyed item is the *Comfortable Air* block.



3. Close the dialog.

Once you're done, your ibd should look like the one below.



i The following figure displays an alternate view of the *VCCU Conceptual Subsystems Communication* ibd. This tutorial doesn't explain how to build the Whitebox ICD Table, but you can find this information in the latest documentation of the modeling tool.

#	Part A	Port A	Port A Features	Item Flow	Port B	Port B Features	Part B
1	Processing	[inout p3 : Internal Data Transfer]	[F] inout inf : Information [F] in dt : Desired Temperature [F] in uc : User Command [F] in at : Ambient Temperature [F] in a : Air	◀ [F] Desired Temperature ▶ [F] Ambient Temperature ▶ [F] Status ▶ [F] User Command ▶ [F] Desired Temperature ▶ [F] in p1 : Manual Control	[F] in p3 : ~Internal Data Transfer [F] in p4 : ~Internal Control [F] in p4 : ~Internal Control [F] in p1 : Manual Control [F] in p2 : Sensor Data Transfer [F] in p3 : Cabin Air Inlet [F] in a : Air	[F] inout inf : Information [F] in cc : CU Command [F] in cc : CU Command [F] in p1 : Manual Control [F] in p2 : Sensor Data Transfer [F] in p3 : Cabin Air Inlet [F] in a : Air	Human-Machine Interacting
2	Processing	[out p4 : Internal Control]	[F] out cc : CU Command	▶ [F] Cool	[F] in p4 : ~Internal Control	[F] in cc : CU Command	Cooling
3	Processing	[out p4 : Internal Control]	[F] out cc : CU Command	▶ [F] Heat	[F] in p4 : ~Internal Control	[F] in cc : CU Command	Heating
4	VCCU	[in p1 : Manual Control]	[F] in dt : Desired Temperature [F] in uc : User Command	▶ [F] User Command ▶ [F] Desired Temperature	[F] in dt : Desired Temperature [F] in uc : User Command [F] in p1 : Manual Control	[F] in dt : Desired Temperature [F] in uc : User Command	Human-Machine Interacting
5	VCCU	[in p2 : Sensor Data Transfer]	[F] in at : Ambient Temperature	▶ [F] Ambient Temperature	[F] in p2 : Sensor Data Transfer	[F] in at : Ambient Temperature	Processing
6	VCCU	[in p3 : Cabin Air Inlet]	[F] in a : Air	▶ [F] Air	[F] in p3 : Cabin Air Inlet	[F] in a : Air	Cooling
7	VCCU	[in p3 : Cabin Air Inlet]	[F] in a : Air	▶ [F] Air	[F] in p3 : Cabin Air Inlet	[F] in a : Air	Heating
8	VCCU	[out p4 : Output Information]	[F] out i : Information	◀ [F] Information	[F] out p4 : Output Information	[F] out i : Information	Human-Machine Interacting
9	VCCU	[out p5 : Cabin Air Outlet]	[F] out ca : Comfortable Air	◀ [F] Cooled Air	[F] out p5 : Cabin Air Outlet	[F] out ca : Comfortable Air	Cooling
10	VCCU	[out p5 : Cabin Air Outlet]	[F] out ca : Comfortable Air	◀ [F] Heated Air	[F] out p5 : Cabin Air Outlet	[F] out ca : Comfortable Air	Heating

Step 8. Allocating functions by conceptual subsystems and synchronizing item flows

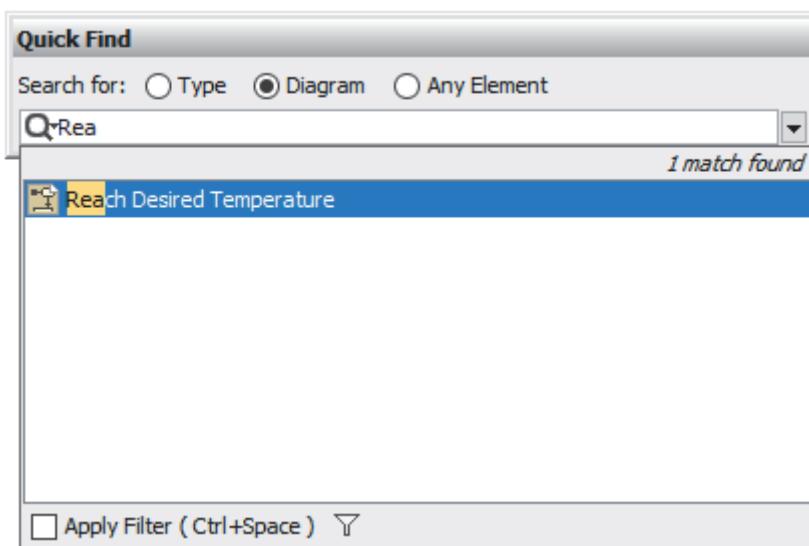
Now it's time to update the *Reach Desired Temperature* diagram (see Chapter [Functional Analysis](#)) with swimlanes, which represent conceptual subsystems of the Vehicle Climate Control Unit. In sequel, actions should be allocated to these swimlanes in order to specify which conceptual subsystems are responsible for performing each function.

These allocations enable you to update the *Reach Desired Temperature* diagram with item flows by synchronizing them from the relevant ibd. This time, we will synchronize from the *VCCU Conceptual Subsystems Communication* diagram.

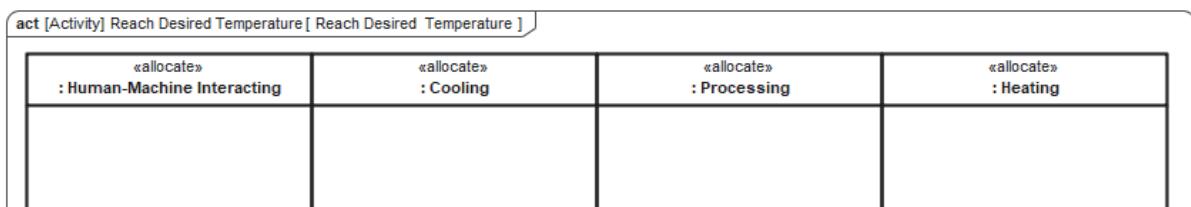
To allocate functions to conceptual subsystems

1. Open the *Reach Desired Temperature* diagram:

- 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 *Reach Desired Temperature* diagram selected in the search results list (see the following figure), press Enter. The diagram is opened.



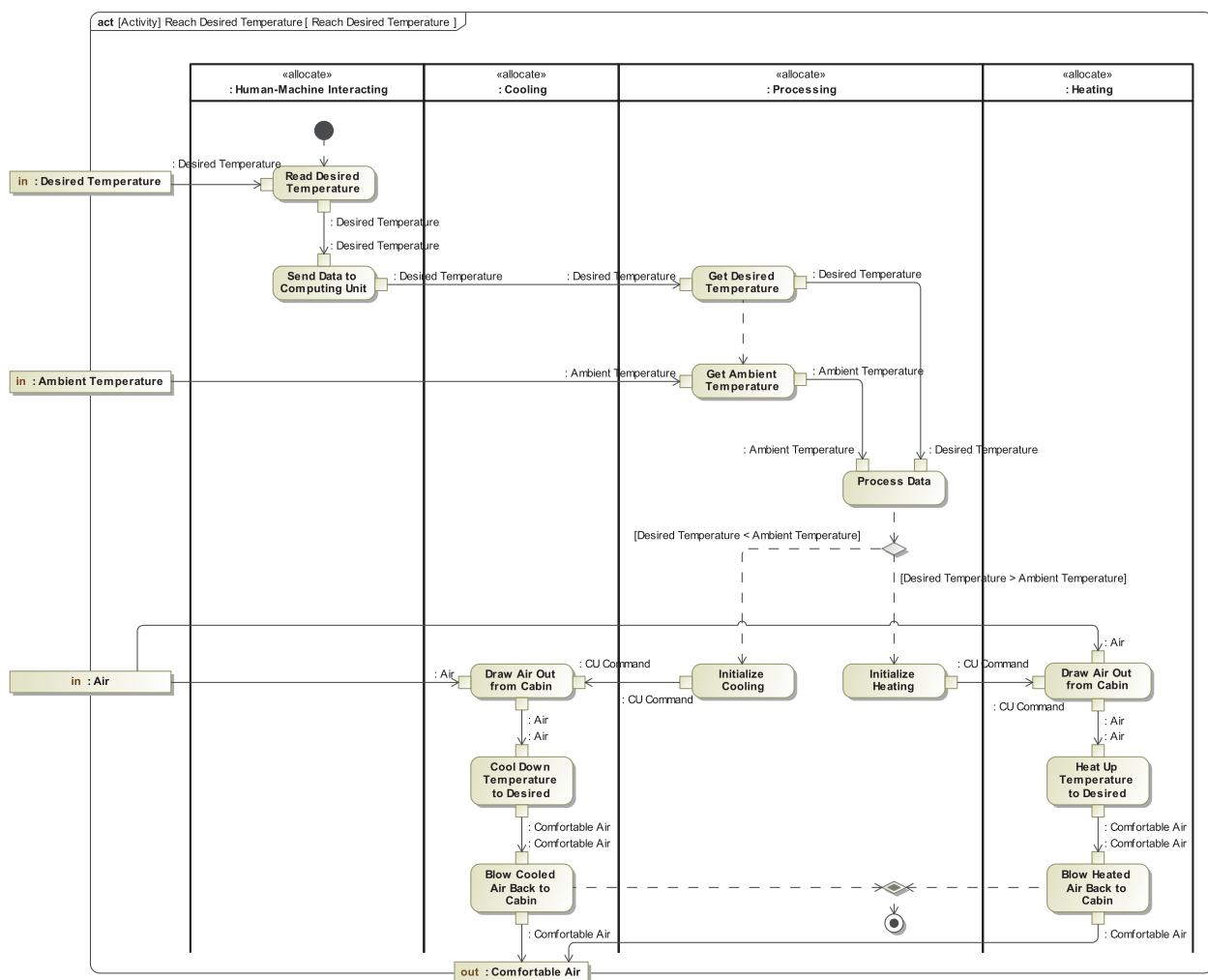
2. Click the **Vertical Swimlanes** button on the activity diagram palette and then click an empty space on that diagram pane. Swimlanes with blank headers are created on the diagram.
3. Right-click any of the swimlanes and select **Insert Swimlane > Vertical to the Left**. A new swimlane with a blank header is created.
4. Repeat step 3 to create another swimlane.
5. In the Model Browser, select the *VCCU* block. For this, use the quick find capability:
 - a. Press Ctrl + Alt + F. The **Quick Find** dialog opens.
 - b. Click **Type** to search for types only and type *VCCU*.
 - c. When you see the *VCCU* block selected in the search results list, press Enter. The *VCCU* block is selected in the Model Browser.
6. Expand the *VCCU* block to see its properties.
7. Select the *:Human-Machine Interacting* part property and drag it onto the header of the first swimlane. The swimlane becomes the representation of that part property.
8. Repeat step 7 for the *:Cooling*, *:Processing*, and *:Heating* part properties in succession. Once you're done, your swimlanes should look like the following figure.



9. Drag the initial node as well as the *Read Desired Temperature* and *Send Data to Computing Unit* actions to the *:Human-Machine Interacting* swimlane.
10. Drag the *Get Desired Temperature*, *Get Ambient Temperature*, *Process Data*, *Initiate Cooling*, and *Initiate Heating* actions, as well as decision, merge, and activity final nodes to the *:Processing* swimlane.
11. Drag one of the *Draw Air Out from Cabin* actions as well as the *Cool Down Temperature to Desired* and *Blow Cooled Air Back to Cabin* actions to the *:Cooling* swimlane.

12. Drag another *Draw Air Out from Cabin* action as well as the *Heat Up Temperature to Desired* and *Blow Heated Air Back to Cabin* actions to the *:Heating* partition.

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

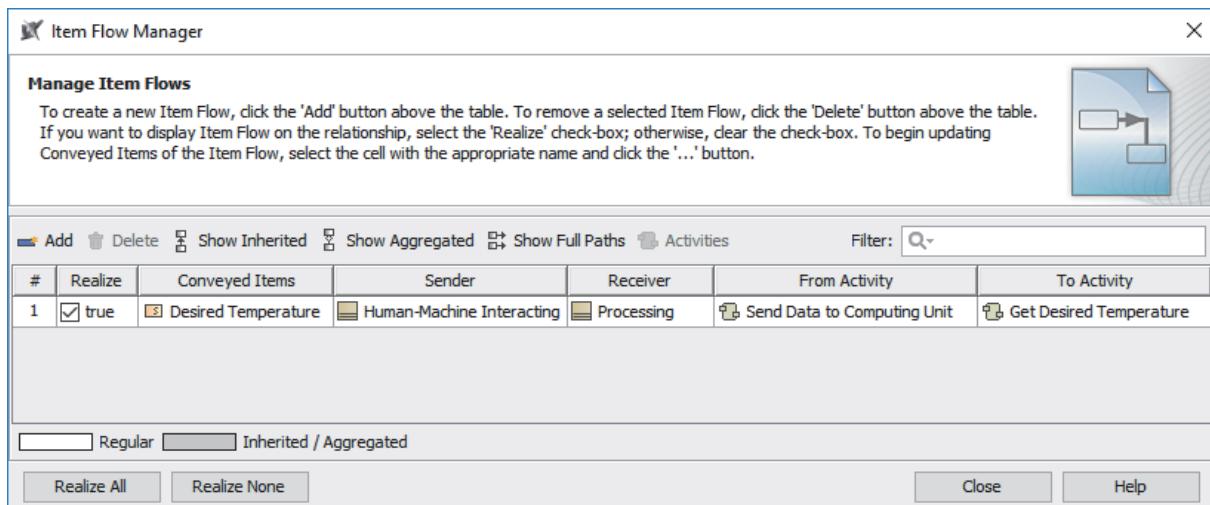


Once all the actions are allocated, it's time to specify item flows among them. Item flows that cross the swimlane boundary can be easily synchronized from the appropriate ibd (in this case the *VCCU Conceptual Subsystems Communication* diagram). Let's start from the item flow between the *Send Data to Computing Unit* and *Get Desired Temperature* actions. The former is allocated to the *:Human-Machine Interacting* part property and the latter is allocated to the *:Processing* part property. If you look at the *VCCU Conceptual Subsystems Communication* diagram, you can see that the *Desired Temperature* signal is flowing between those part properties. Therefore, you can conclude that the same signal is flowing between the actions they allocate. The **modeling tool** helps you specify the item flows without switching between diagrams, as outlined below.

To synchronize the item flow between the *Send Data to Computing Unit* and *Get Desired Temperature* actions from the relevant ibd

1. Select the object flow between these actions.
2. Click the Item Flow Manager button  on its smart manipulator toolbar.
3. In the open dialog, click to select the only item flow in the list. It conveys the *Desired Temperature* signal.

 The list always includes the item flows that are possible between two actions. These item flows are taken from the relevant ibd.

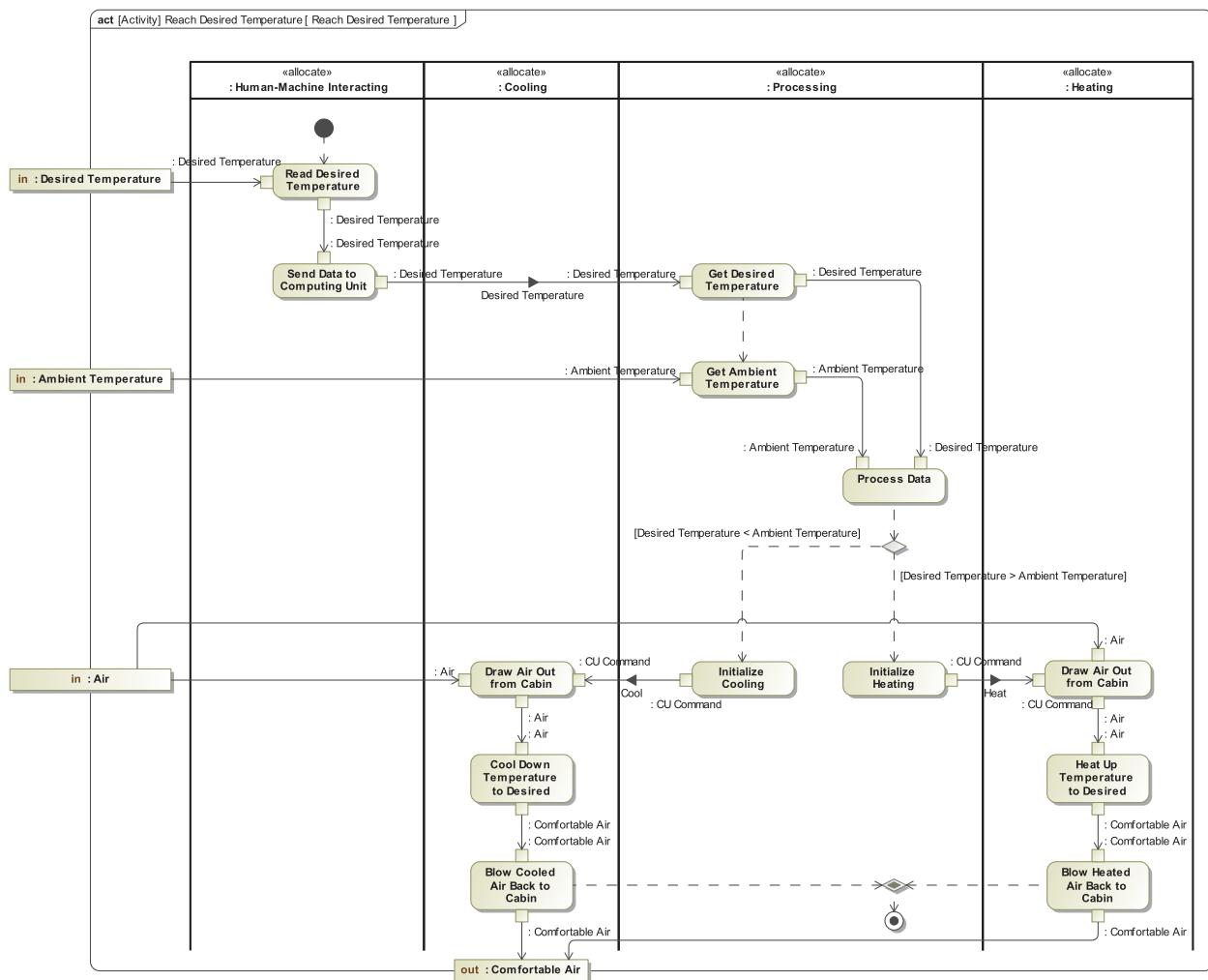


4. Close the dialog. As a result, the item flow conveying *Desired Temperature* signal displays in the diagram.

In the same manner, specify that:

- The *Cool* signal is flowing from the *Initialize Cooling* action to the *Draw Air Out from Cabin* action.
- The *Heat* signal is flowing from the *Initialize Heating* action to the *Draw Air Out from Cabin* action.

After you're done, your diagram should look very similar to the one below.



Next you can specify the item flows within the same swimlane, starting from the item flow between the *Read Desired Temperature* and *Send Data to Computing Unit* actions.

To specify the item flow conveying the *Desired Temperature* signal between the *Read Desired Temperature* and *Send Data to Computing Unit* actions

1. Select the object flow between these actions.
2. Click the New Item Flow button  on its smart manipulator toolbar.
3. In the open dialog, make sure that the *Desired Temperature* signal is selected as the conveyed item and click **Finish** (see the following figure). As a result, the *Desired Temperature* signal becomes conveyed by the item flow between the relevant actions.

Create / Edit Item Flow

New or Existing Item Flow

Create a new or edit an existing Item Flow by specifying the conveyed items and direction. Synchronize this Item Flow in the related diagrams by checking the "Add Item Flow to corresponding relationships" field.

Realize the Item Flow on the activity edge between the selected activities in Step 2

1. Create new or Select Existing Item Flow

2. Specify Activities

Item Flow: <NEW>

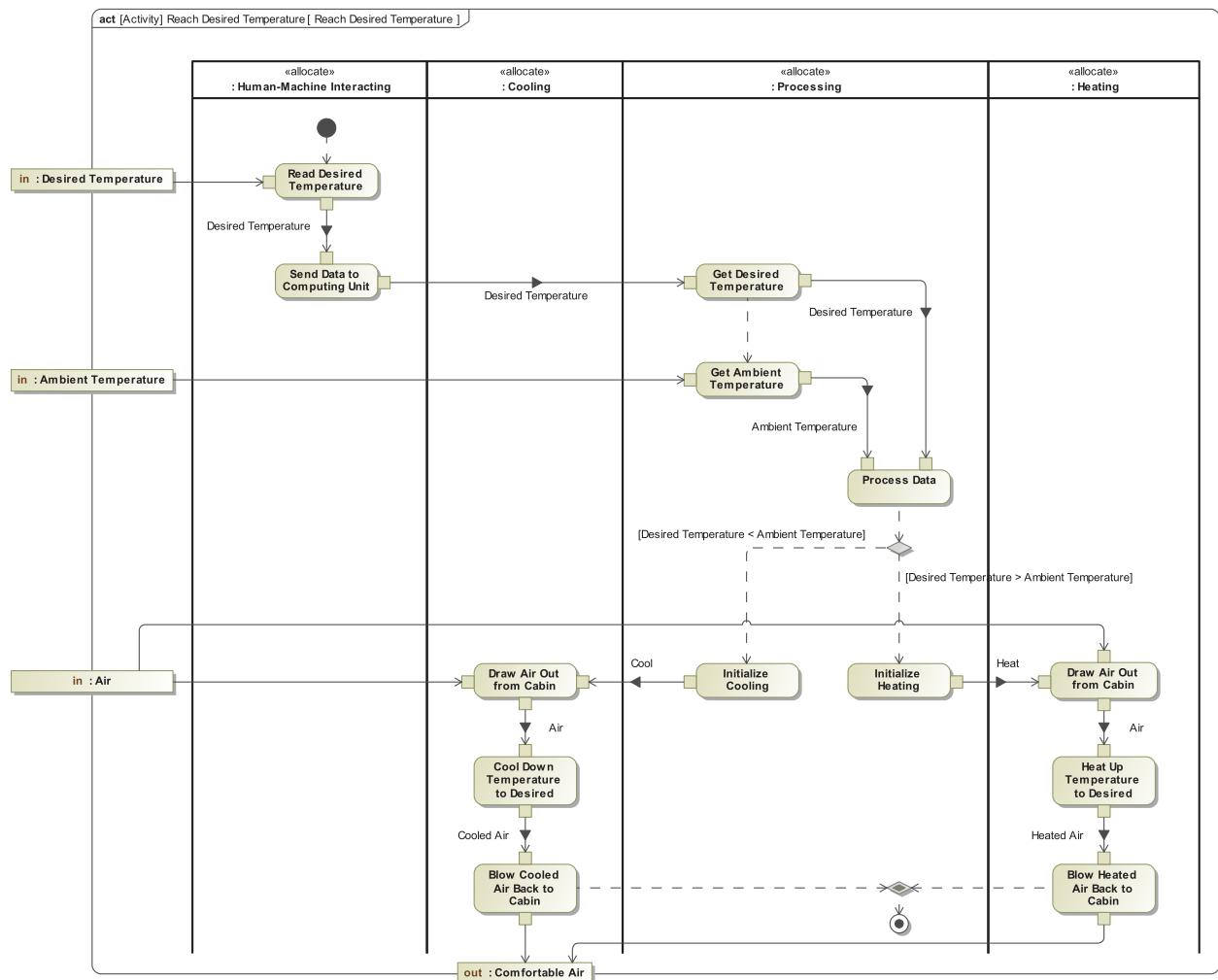
Direction: From Human-Machine Interacting To Human-Machine Interacting

Conveyed Items: Desired Temperature

Add Item Flow to corresponding relationships

< Back Next > Finish Cancel Help

Following the previous procedure, specify the rest of the item flows you see in the figure below. Keep in mind that you can be more specific when specifying the item flow between the *Cool Down Temperature to Desired* and *Blow Cooled Air Back to Cabin* actions. For this, you should select the *Cooled Air* block instead of the *Comfortable Air* block in the **Create / Edit Item Flow** dialog. Remember that you can also be more specific when specifying the item flow between the *Heat Up Temperature to Desired* and *Blow Heated Air Back to Cabin* actions.



- i** Pins don't display their types; only item flows are shown on the diagram. This makes the diagram less overcrowded and easier to read. If you want to get rid of pin types, do the following:
1. Press the Alt key and select any input pin. All input pins are selected on the diagram.
 2. Right-click the selection and click **Show Type** to clear the selection.
 3. Repeat the steps above to hide the output pin types.

Although more detailed than the use case scenario (see Chapter [Use Cases](#)), this one is still a high-level scenario and may require decomposition as well. Further structural decomposition stimulated by the functional decomposition is performed in parallel. The functional and structural breakdown may have as many iterations as you need to understand the problem domain.

Conceptual Subsystems done. What's next?

- System functions and its conceptual subsystems, with or without quantifiable characteristics of the **Sol**, refine stakeholder needs. Therefore, once you have finished it, you can establish the traceability relationships to stakeholder needs (see Chapter [Traceability to Stakeholder Needs](#)).
- You might also need to specify quantifiable characteristics for one or more conceptual subsystems, which would be more specific than those defined for the whole **Sol** (see Chapter [Measures of Effectiveness](#)). In that case, you should move to the **MoEs for Subsystems** cell.

MoEs for Subsystems

What is it?

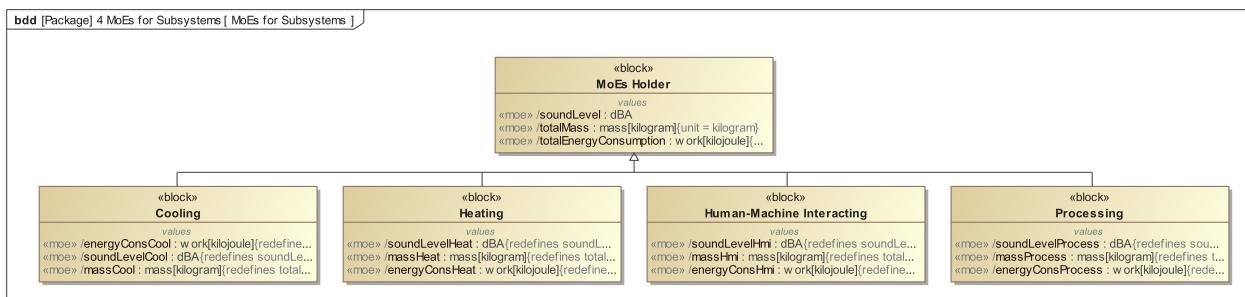
In this cell, you should specify the **MoEs** for one or more conceptual subsystems (aka functional blocks) of the **Sol** to make further refinements of non-functional **stakeholder needs**. This cell is optional, as you might not need to specify MoEs in addition to those defined for the entire **Sol** (see Chapter **Measures of Effectiveness**).

Who is responsible?

MoEs for conceptual subsystems can be identified by the Systems Analyst or Systems Engineer.

How to model?

To capture MoEs for conceptual subsystems in the **modeling tool**, you can utilize the infrastructure of the SysML **bdd** – the same way you used it for capturing MoEs of the **Sol** in the **Measures of Effectiveness** cell. Just like in that cell, measures of effectiveness for a conceptual subsystem can be captured in the model as value properties of that subsystem, with the «moe» stereotype applied.

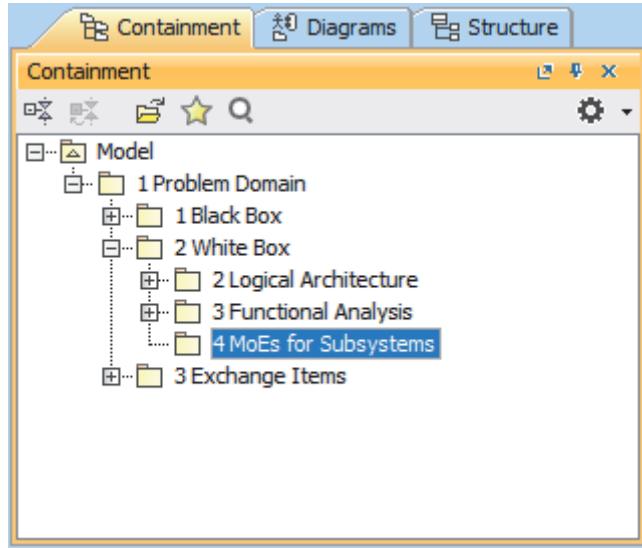


Tutorial

- **Step 1. Organizing the model for MoEs for subsystems**
- **Step 2. Capturing MoEs for subsystems**

Step 1. Organizing the model for MoEs for subsystems

Following the structure of the MagicGrid framework, model elements that capture MoEs for subsystems should be stored in a separate package inside the *2 White Box* package. We recommend naming the package after the cell, that is, *4 MoEs for Subsystems*.



To organize the model for MoEs for subsystems

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 *4 MoEs for Subsystems* to specify the name of the new package and press Enter.

Step 2. Capturing MoEs for subsystems

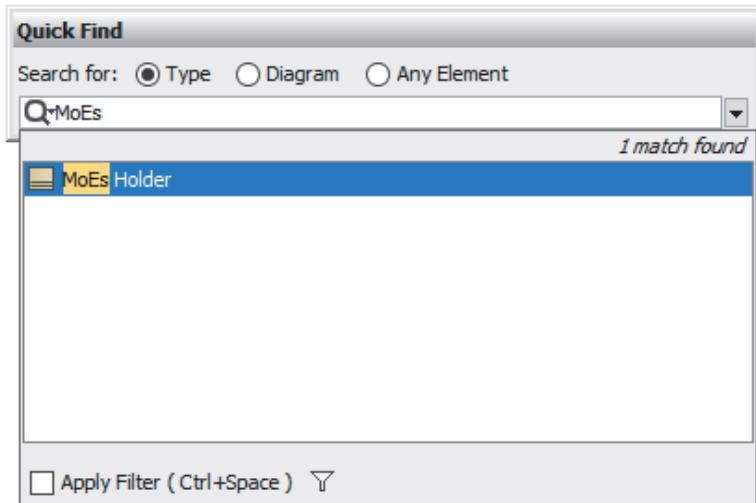
Let's specify the MoEs for the conceptual subsystem of the VCCU, captured in the model as the *Cooling* block. Just like MoEs of the entire **Sol**, MoEs of that conceptual subsystem refine the same non-functional stakeholder needs: *SN-1.1.1*, *SN-1.1.4*, and *SN-1.2.1*. Therefore, the *Cooling* block can inherit all the MoEs from the same MoEs holder introduced in the **Measures of Effectiveness** cell. Later, the inherited value properties can be redefined if there is a need to change their properties (such as the name or type). In this case, we will change only names, as types are relevant.

To capture MoEs for the Cooling subsystem

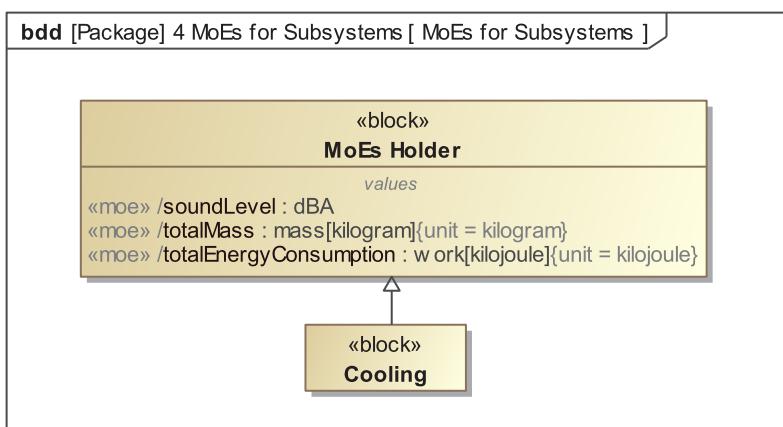
1. Create a bdd. For this:
 - a. Right-click the *4 MoEs for Subsystems* package and select **Create Diagram**.
 - b. In the search box, type *bdd* (the acronym of the SysML block definition diagram), and then double-press Enter. The diagram is created.

(i) Note that the diagram is named after the package where it is stored. This name perfectly suits the diagram, too. You need only remove the sequence number from its name.

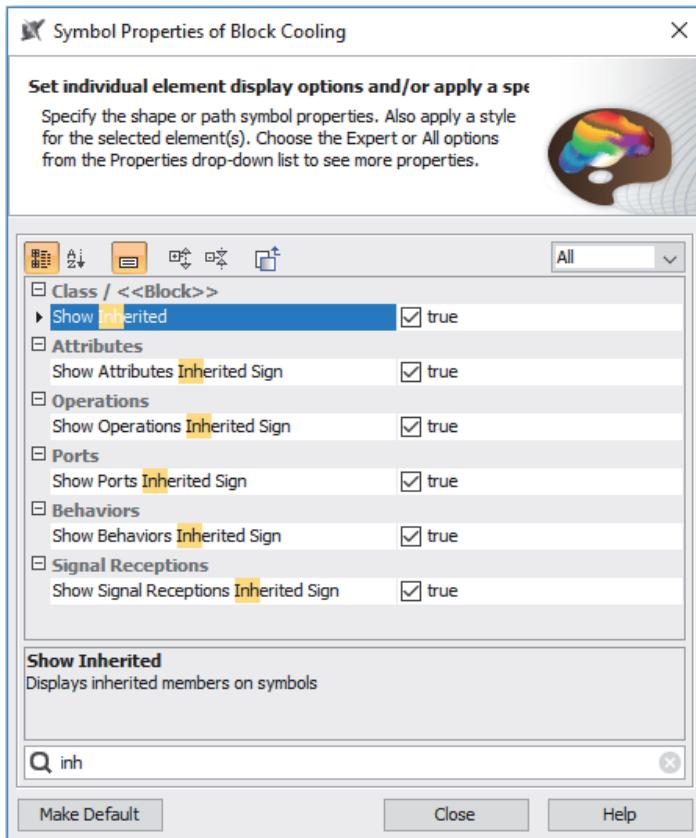
2. Display the *MoEs Holder* block on the newly created diagram. For this:
 - a. Press **Ctrl + Alt + F**. The **Quick Find** dialog opens.
 - b. Type *MoEs*.
 - c. When you see the *MoEs Holder* block selected in the search results list (see the following figure), press Enter. The *MoEs Holder* block is selected in the Model Browser.



- d. Drag the block to the diagram pane.
3. In the same way, display the *Cooling* block on the diagram as well.
 4. Select the *Cooling* block and click the Generalization button on its smart manipulator toolbar.
 5. Select the *MoEs Holder* block. As a result, the *Cooling* block inherits all the properties of the *MoEs Holder* block.



6. Display inherited properties on the shape of the *Cooling* block. For this:
 - a. Right-click the block and select **Symbol Properties**.
 - b. On the top-right corner of the open dialog, select **All** from the drop-down list to switch the dialog to the All Properties mode.
 - c. In the search box below the properties list, type *inh*.
 - d. When you see the **Show Inherited** symbol property displayed on the top of the list, click its value cell to set it to true.



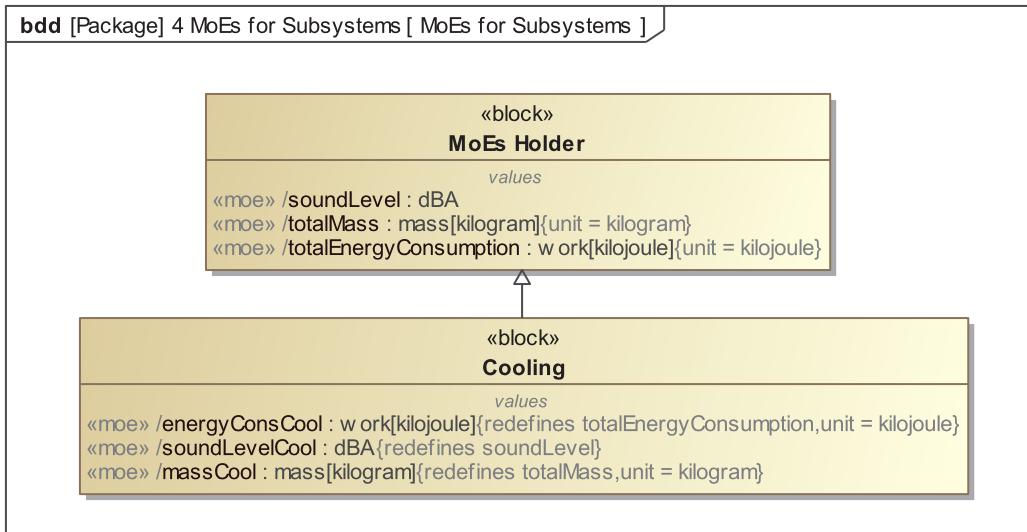
- e. Close the dialog.
7. Redefine the *totalEnergyConsumption* value property. For this:
- Double-click the *Cooling* block to open its Specification.
 - On the left side of the dialog, click **Properties**.
 - On the right side, select the *totalEnergyConsumption* value property and click the **Redefine** button below.
 - In the **Name** cell, type the new name of the property, for example, *energyConsCool*.
 - Press Enter to confirm the changes.

Properties				
	Name	Type	Default Value	Owner
Value Properties				
	energyConsCool	work[kilojoule] [ISO-80...		Cooling [1 Problem...]
^	soundLevel	dB [1 Problem Domain...]		MoEs Holder [1 Pr...]
^	totalMass	mass[kilogram] [ISO-8...		MoEs Holder [1 Pr...]

Up **Down** **Create** **Redefine** **Delete**

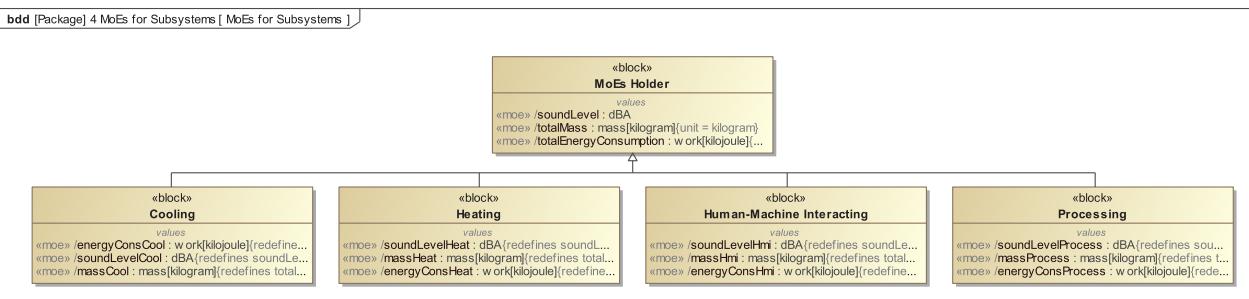
8. In the same way, redefine other properties.
 9. Close the dialog.

When you're finished, your diagram should look like the following figure.



After you capture the MoEs for all conceptual subsystems of the Vehicle Climate Control Unit, your diagram should look similar to the one below.

- i** As you can see, the *values* compartments are cut to display the full length of value property rows. For this, do the following:
1. Select all the blocks that capture conceptual subsystems of the Vehicle Climate Control Unit. Press Shift for multiple selection.
 2. Right-click the selection and choose **Symbol Properties**.
 3. In the open dialog, find the **Shrinkable Attributes and Operations Compartment** property and set its value to *true*.



MoEs for Subsystems done. What's next?

When you have the MoEs for subsystems, you can specify what stakeholder needs they refine (see Chapter [Traceability to Stakeholder Needs](#)).

Traceability to Stakeholder Needs

What is it?

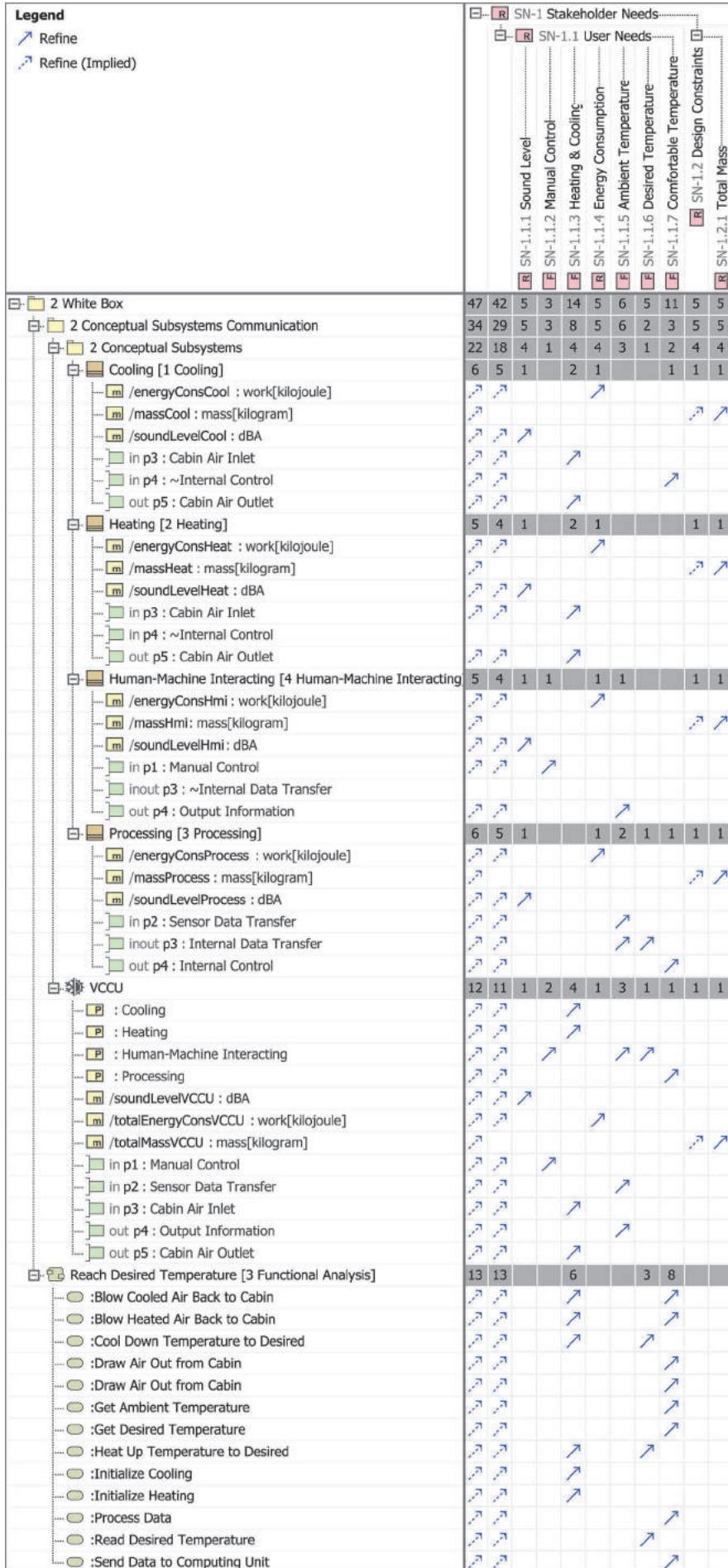
In order to have a complete problem domain model, you need to establish traceability relationships between the elements that capture stakeholder needs and the elements that capture the white-box functions, conceptual subsystems, conceptual interfaces, and non-functional characteristics of the [Sol](#). 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, such as actions that capture leaf functions of the functional decomposition model. When you're done with traceability relationships here, every item of the stakeholder needs must be refined by one or more elements of the other SysML types. The opposite means that not all stakeholder needs are considered in the problem domain model, which may lead to incorrect system requirements specification (see Chapter [System Requirements](#)), resulting in developing the wrong system.

Who is responsible?

Traceability relationships can be established by the Systems Analyst or Systems Engineer.

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.

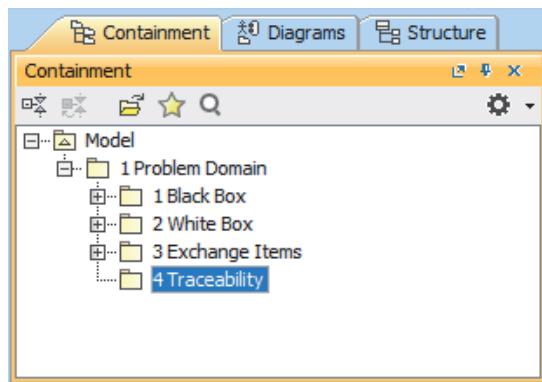


Tutorial

- Step 1. Organizing the model for traceability
- Step 2. Creating a matrix for capturing refine relationships
- Step 3. Capturing refine relationships

Step 1. Organizing the model for traceability

Traceability relationships and the views where they are displayed can be stored in a separate package, which is created directly under the *1 Problem Domain* package as the last of its internal packages to represent the modeling workflow.



To organize the model for traceability

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 *4 Traceability* to specify the name of the new package and press Enter.

Step 2. Creating a matrix for capturing refine relationships

To begin capturing refine relationships, you should create a Refine Requirement Matrix first. The matrix is predefined to represent requirements in its columns and refine relationships in its cells. You only need to specify that the rows of the matrix display the following:

- Call behavior actions, which capture functions of the **Sol**
- Part properties, which capture conceptual subsystems of the **Sol**
- Proxy ports, which capture the interfaces of the **Sol**
- Value properties with the «moe» stereotype applied as they capture MoEs of the **Sol** or its subsystems

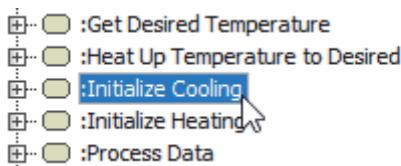
Remember, stakeholder needs must be refined by the problem domain elements from the lowest level of detail. Therefore, the column elements must be taken from the package that contains the white-box analysis results.

To create a Refine Requirement Matrix

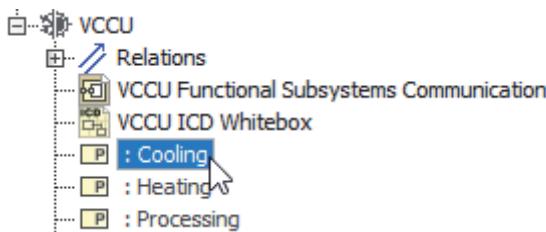
1. Right-click the *4 Traceability* package and select **Create Diagram**.
2. In the open dialog, click the **Expert** button to turn on the appropriate mode.
3. In the search box, type *rrm*, the acronym of the Refine Requirement Matrix, and then press Enter. The matrix is created.
4. Type *Refine Stakeholder Needs* to specify the name of the new matrix and press Enter again.
5. Specify the column element scope:

- a. In the Model Browser, select the *SN-1 Stakeholder Needs* requirement.
- 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.
6. Specify row element types:

- a. In the Model Browser, select any call behavior action, such as the one typed by the *Initialize Cooling* activity (which appears under the *Reach Desired Temperature* activity).



- b. 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 call behavior action type.
- c. In the Model Browser, select any part property; for example, the one typed by the *Cooling* block.

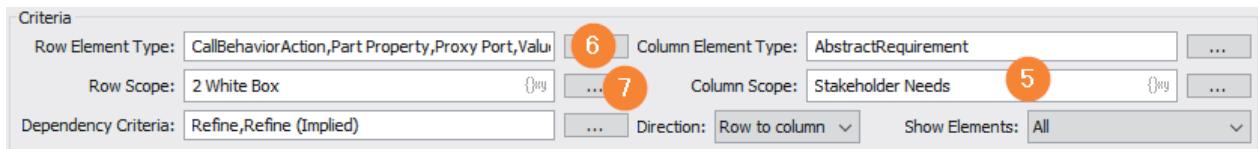


- d. Hold down the Ctrl key and drag the selection onto the **Row Element Type** box in the **Criteria** area above the matrix contents. The rows of the matrix are set to display the elements of the part property type as well.
- e. Repeat steps c and d to enable the rows of the matrix to display the proxy ports and value properties with the «moe» stereotype.

7. Specify the row element scope:

- a. In the Model Browser, select the *2 White Box* package.
- b. Drag the selection onto the **Row Scope** box in the **Criteria** area above the matrix contents. The selected package becomes the scope of the matrix rows.

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



- i** If the matrix displays interface blocks, you can easily get rid of them by modifying its display settings. For this, click the ... button next to the **Row Element Type** box and in the open dialog, click to clear the **Include Subtypes** check box. Don't forget to refresh the matrix.

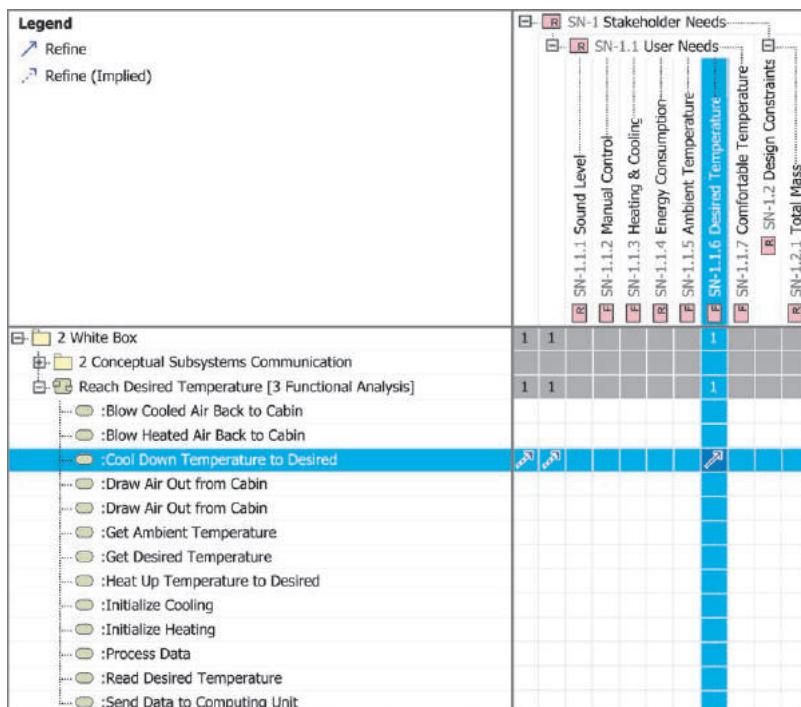
Step 3. Capturing refine relationships

Now you're ready to capture refine relationships. Let's create the one from the *Cool Down Temperature to Desired* action to the *SN-1.1.6 Desired Temperature* stakeholder need (captured as a requirement) to convey that the action refines the need.

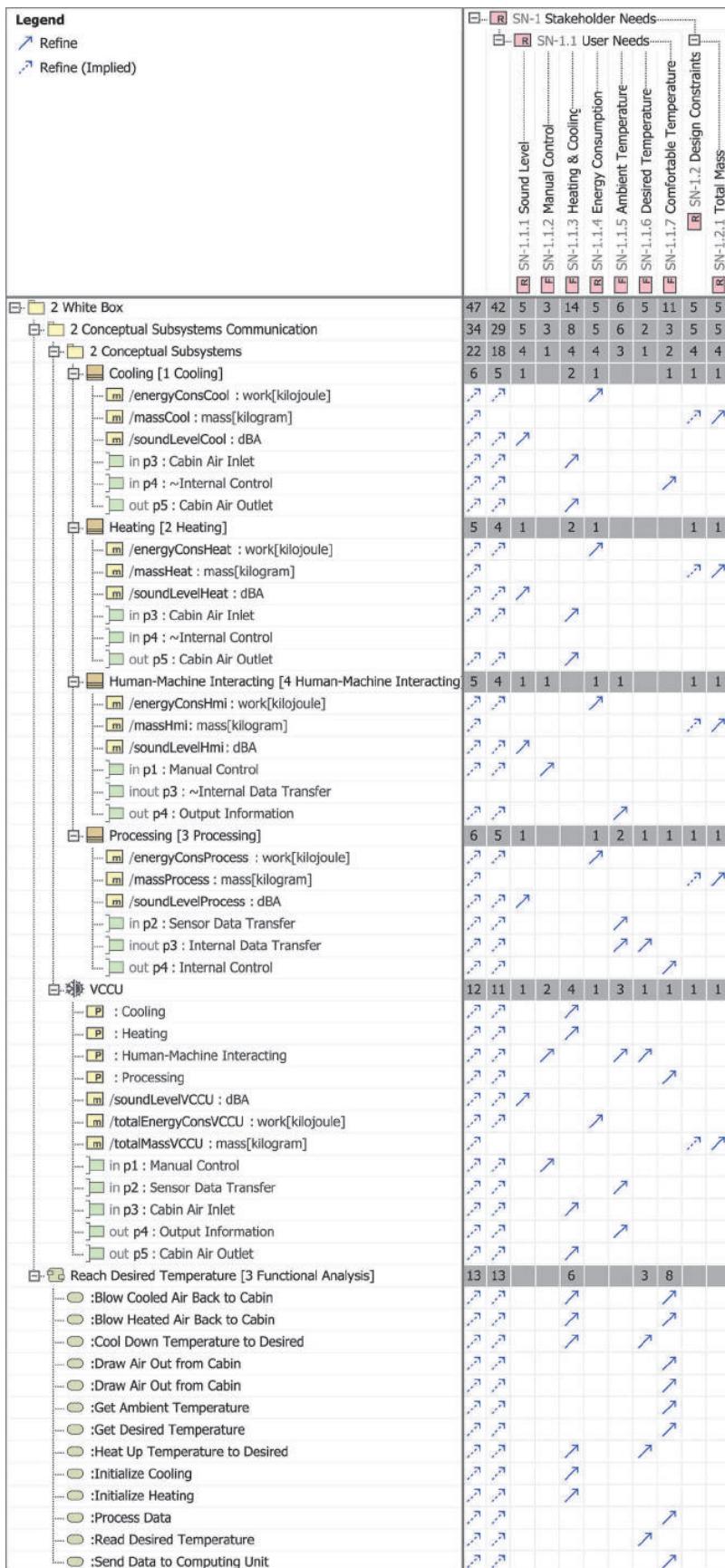
To specify the refine relationship in the matrix

- Double-click the cell at the intersection of the row that displays the *Cool Down Temperature to Desired* action and the column that displays the *SN-1.1.6 Desired Temperature* requirement. The derivation relationship is created between the appropriate items and displayed in the cell.

- i** To save space, the contents of the *2 Conceptual Subsystems Communication* package were hidden in the following figure.



After all refine relationships are captured, your *Refine Stakeholder Needs* matrix should look like the one in the following figure. Remember that when you're done, every item of the stakeholder needs must be refined by one or more elements from the problem domain analysis model.



As you might have noticed, the matrix displays implied refine relationships (dotted arrows) as well. Once the leaf requirement is refined, all upper level requirements of the same branch are also considered refined; this is the logic of establishing implied refine relationships. For information on how to create and use implied relationships, refer to the latest documentation of the [modeling tool](#).

Problem Domain done. What's next?

The problem domain analysis is complete when at least 90 percent of stakeholder needs (this threshold may vary in different organizations) are refined by one or more elements from the problem domain model. For monitoring the status of the problem domain analysis project, you can utilize the metrics feature supported by the [modeling tool](#). The metric table in the following figure calculates the metrics specified by the **Requirement Refinement** metric suite. For more information about using metrics, see the latest documentation of the [modeling tool](#).

#	Date	Scope	Requirements	Refined Requirements	Refined Requirements Percentage
1	2020.01.05 10:00	1 Stakeholder Needs	11	0	0
2	2020.01.10 14:00	1 Stakeholder Needs	11	6	55
3	2020.01.12 11:25	1 Stakeholder Needs	11	9	82
4	2020.01.14 16:45	1 Stakeholder Needs	11	11	100

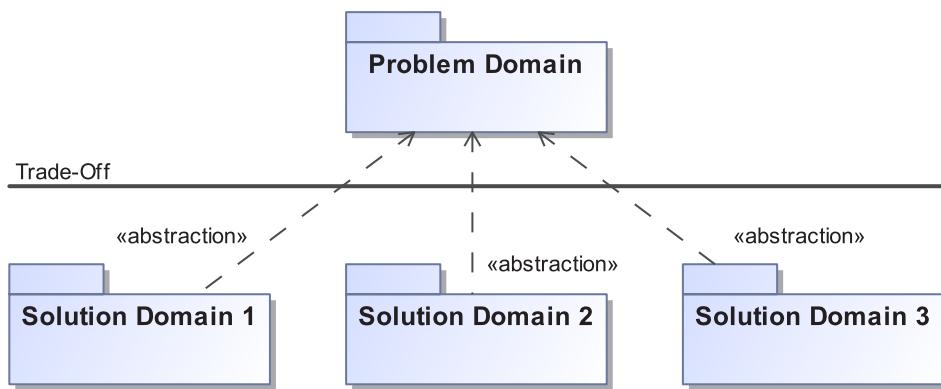
The complete problem domain analysis model, including the stakeholder needs for the **Sol**, the expected functions and conceptual structure of the **Sol**, either with or without the MoEs, can be considered as initial system requirements specification in the model-based form. If the system requirements specification is required in textual form (which is not a rare case), the problem domain analysis model can be transformed (either manually or automatically by utilizing the transformation algorithms) into the SysML requirements model and provided to system architects as input for developing the logical architecture of the **Sol** in the solution domain. This is described in Chapter [Solution domain](#).

SOLUTION DOMAIN

Once the problem domain analysis is completed and the stakeholder needs for the **Sol** are transferred into the SysML model, it's time to start thinking about the solution domain model of that **Sol**. The solution domain model defines a precise cross-discipline logical architecture of the **Sol** to solve the problem defined in the problem domain analysis model. It should not be mixed with detailed design models, which are created in later phases of the **Sol** development and are not part of the MagicGrid framework.

As the same problem definition can have several solutions, a trade-off analysis can be performed to choose the optimal solution for implementing the system (see the following figure).

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



As you can see in the following table, creating the solution domain model consists of specifying requirements, structure, behavior, and parameters of the **Sol**. Furthermore, the task of building a logical architecture usually includes 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. The precision of the system architecture model depends on the number of iterations.

	Pillar					
Domain	Problem	Requirements	Structure	Behavior	Parameters	Safety & Reliability
		Stakeholder Needs	System Context	Use Cases	Measures of Effectiveness (MoEs)	Conceptual and Functional Failure Mode & Effects Analysis (FMEA)
	Solution	Conceptual Subsystems	Functional Analysis	MoEs for Subsystems	Conceptual Subsystems FMEA	
		System Requirements	System Structure	System Behavior	System Parameters	System Safety & Reliability (S&R)
		Subsystem Requirements	Subsystem Structure	Subsystem Behavior	Subsystem Parameters	Subsystem S&R
	Implementation	Component Requirements	Component Structure	Component Behavior	Component Parameters	Component S&R
	Implementation	Implementation Requirements				

Deploying the solution domain

The logical architecture of the **Sol** 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 domain model exceeds the scope and complexity of the problem domain definition, especially when it includes more than one solution. Specifying the solution domain in two or more separate models helps to manage the information 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 a method of deploying the solution domain across multiple models.

As stated before, more than one solution can be provided for the same problem. To explain our approach, we introduce the case of three solution domain models: *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 imagine them similar to the contents of the *Solution Domain 3*. The *Logical System Architecture (LSA)* model is the core of each solution domain. Based on the results of the problem domain analysis, it defines the logical subsystems of the **Sol** in a single-level hierarchy. As you can see in the following figure, there are three logical subsystems in this case: *Subsystem 1*, *Subsystem 2*, and *Subsystem 3*. It's important to note that each subsystem is considered a black box in the *LSA* model, which also defines logical interfaces for each subsystem to determine how they shall operate with one another and integrate into the whole according to system requirements. Interfaces are not included in the figure below, to avoid clutter. The owner of the *LSA* model is the systems architect, who has "a big picture view" of the **Sol**. In other words, the systems architect is like the conductor of an orchestra: he/she ensures everyone is playing the same melody at the same time.

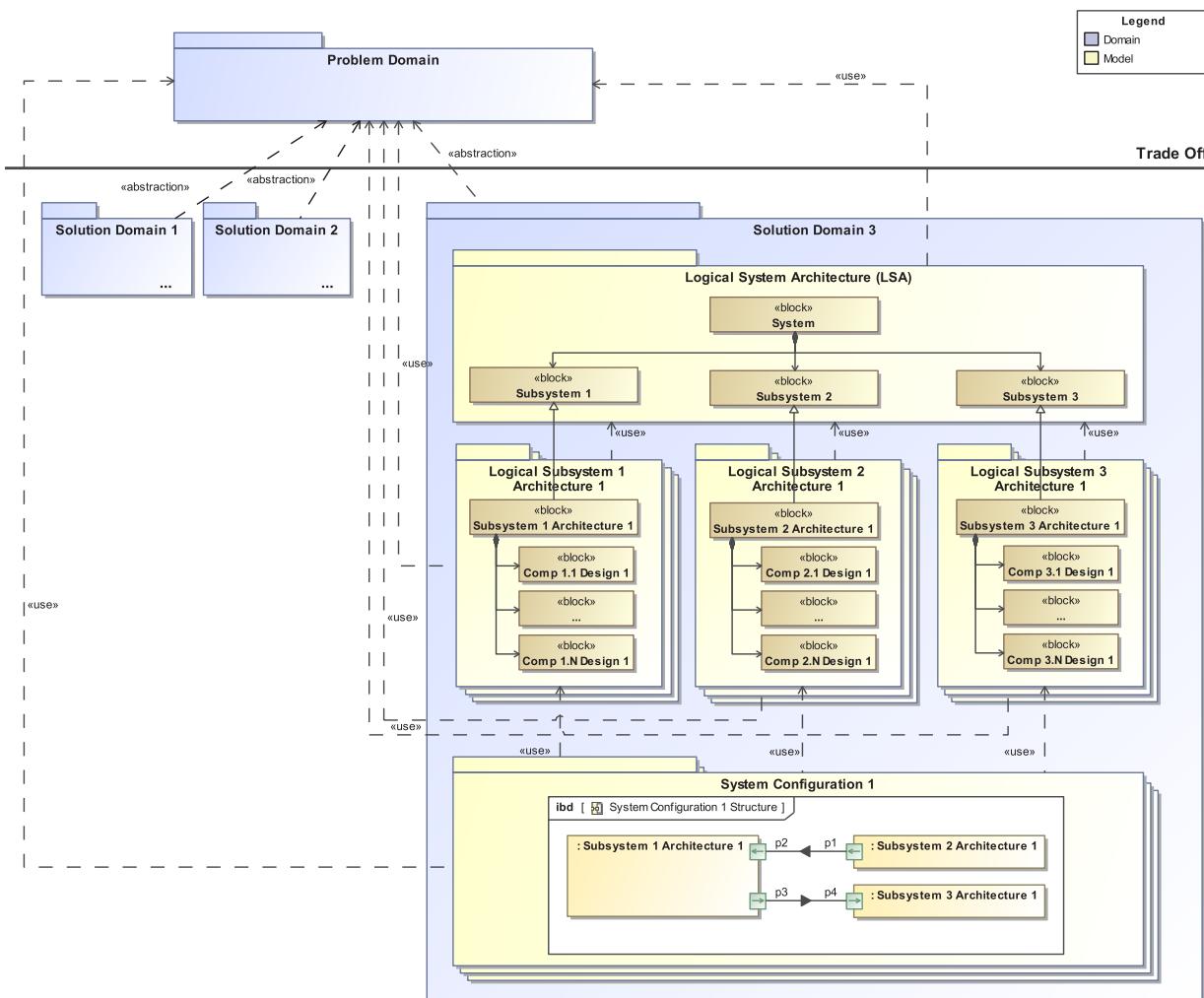
Definition of logical subsystems in the *LSA* model helps to identify work packages and appoint them to separate engineering teams. Every team develops one or more logical architectures for the appointed subsystem. It may happen that they work in parallel. As seen in the following figure, each of the three subsystems has several solutions. *Logical Subsystem X Architecture 1*, where *X* stands for 1, 2, or 3, is visible on the top and the others are hidden underneath. The generalization relationship (solid line with hollow, triangular arrowhead) between the *Subsystem 1* and *Subsystem 1 Architecture 1* blocks means that the engineering team working on the *Logical Subsystem 1 Architecture 1* model knows (or, in terms of SysML, inherits) the design constraints (such as logical interfaces) of the logical subsystem defined in the *LSA* model, and must follow them.

Other generalizations convey appropriate information. Architectures of different logical subsystems and even different architectures of the same logical subsystem may include a diverse number of components. A solution domain model of a logical subsystem defines the requirements, structure, behavior, and parameters of that logical subsystem.

After engineering teams finish with the logical architectures of all subsystems defined in the *LSA* model, the systems architect can integrate them into a single model. He/she picks up the preferred one of each logical subsystem and builds the integrated logical architecture of the whole system. Note that 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 logical architecture of every subsystem, the integrated logical architecture of the whole **Sol** defines the structure, behavior, and parameters, and must meet system requirements specification.

Having the full model of the **Sol**, the systems architect can perform a trade-off analysis and pick up the optimal system configuration; having optimal system configurations of several solution domains, the systems architect can pick up one optimal solution above all. This is illustrated by the *Trade Off* analysis in the following figure.

It is important to mention that trade offs can be done not only between solution domains, but at every level of granularity in the logical architecture of the **Sol**.



Re-applying MagicGrid from scratch in the solution domain

It commonly happens that subsystem requirements are not adequate to build the logical architecture of the subsystem, especially when it is developed by a supplier (contractor), as the Original Equipment Manufacturer (OEM, contracting authority) usually is not willing to share the problem domain model. Thus, the supplier may decide to perform the problem domain analysis of that logical subsystem beforehand. The figure below illustrates re-applying the MagicGrid framework from scratch, at the logical subsystem level, where the logical subsystem is considered the **Sol**.

MagicGrid at System Level

Pillar					
Domain	Methodology	Requirements	Structure	Behavior	Parameters
		Stakeholder Needs	System Context	User Cases	Maturity of Effectiveness (MoE)
Initial	BB&S	Conceptual Subsystems	Functional Analysis	MoEs for Subsystems	Conceptual Subsystems PFA
System	System	System Requirements	System Structure	System Behavior	System Parameters
Subsystem	Subsystem	Subsystem Requirements	Subsystem Structure	Subsystem Behavior	Subsystem Parameters
Component	Component	Component Requirements	Component Structure	Component Behavior	Component Parameters
Implementation	Implementation	Implementation Requirements			Implementation S&R

MagicGrid at Subsystem/Component Level

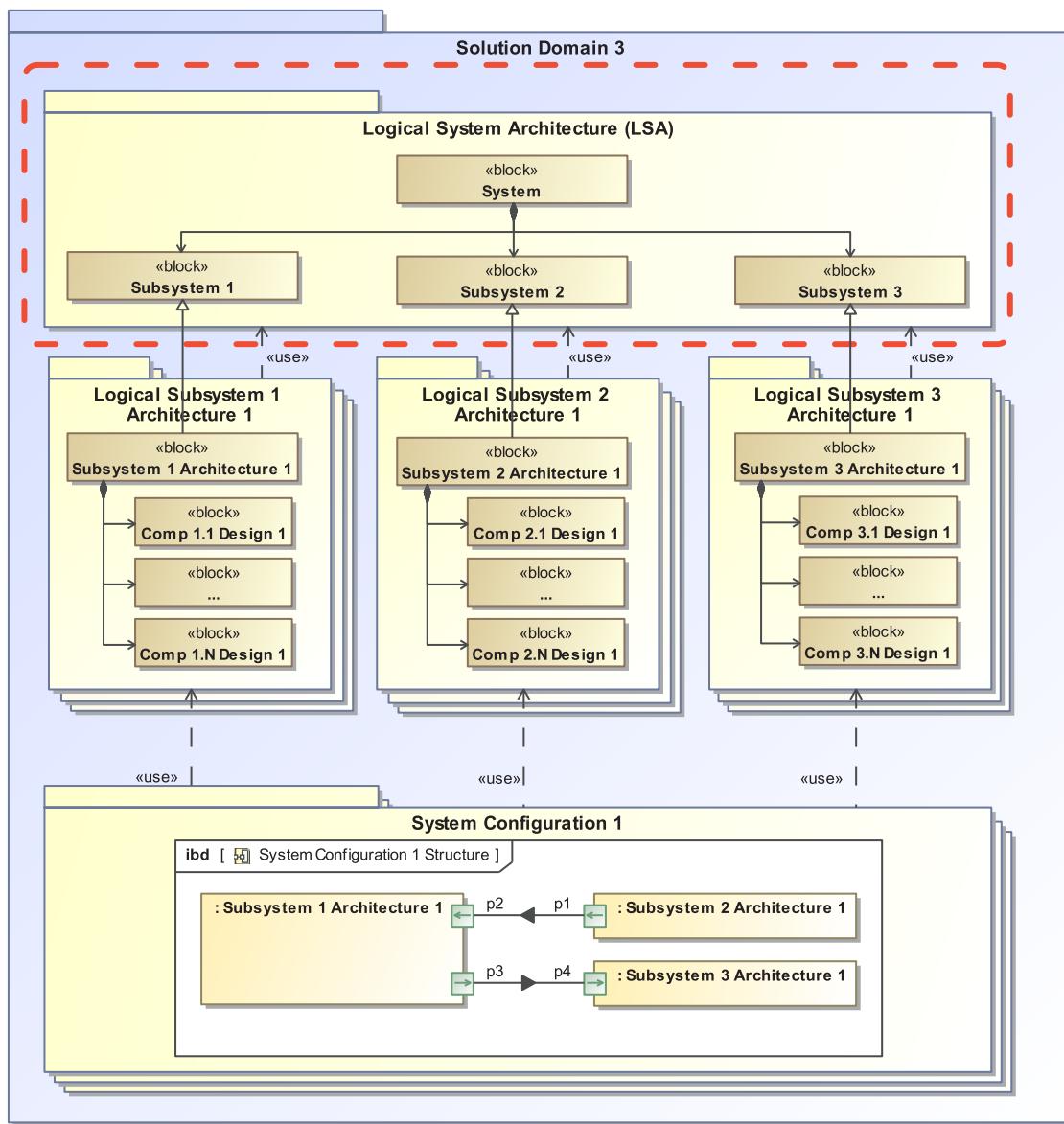
Pillar					
Domain	Methodology	Requirements	Structure	Behavior	Parameters
		Stakeholder Needs	System Context	User Cases	Maturity of Effectiveness (MoE)
Domain	BB&S	Conceptual Subsystems	Functional Analysis	MoEs for Subsystems	Conceptual Subsystems PFA
Subsystem	Subsystem	Subsystem Requirements	Subsystem Structure	Subsystem Behavior	Subsystem Parameters
Component	Component	Component Requirements	Component Structure	Component Behavior	Component Parameters
Implementation	Implementation	Implementation Requirements			Implementation S&R

Subsystem 1 (Supplier 1)
Subsystem 2 (Supplier 2)
Subsystem 3 (Supplier 3)

Building the logical architecture of the Sol

In this phase of specifying the solution domain, you should imagine yourself being the systems architect and working to deliver the logical architecture of the **Sol** (i.e., building the logical system architecture (LSA) model). The essential goal of building this model is to define the logical subsystems of the **Sol**, which leads to identification of work packages.

As mentioned before, the logical architecture of the **Sol** can be stored in a separate model, apart from the problem domain definition. The highlighted area in the following figure shows exactly which part of the solution domain we are talking about.



As you may conclude looking at the design layout of the MagicGrid framework (see the highlighted area in the following figure), the system-level solution domain model specifies the system requirements, structure, behavior, and parameters. Additionally, every SysML element you create in the LSA model must satisfy at least one system requirement. Otherwise, there is no motivation for having it in the LSA model. Therefore, you have to start and end the system-level solution domain model with system requirements: first specify them and, when you have the entire LSA model, establish traceability relationships between them and logical system architecture elements.

		Pillar				
Domain	Problem	Requirements	Structure	Behavior	Parameters	Safety & Reliability
		Stakeholder Needs	System Context	Use Cases	Measures of Effectiveness (MoEs)	Conceptual and Functional Failure Mode & Effects Analysis (FMEA)
	Solution	Conceptual Subsystems	Functional Analysis	MoEs for Subsystems	Conceptual Subsystems FMEA	
		System Requirements	System Structure	System Behavior	System Parameters	System Safety & Reliability (S&R)
		Subsystem Requirements	Subsystem Structure	Subsystem Behavior	Subsystem Parameters	Subsystem S&R
	Implementation	Component Requirements	Component Structure	Component Behavior	Component Parameters	Component S&R
	Implementation	Implementation Requirements				

System Requirements

What is it?

To specify a logical system architecture, you first need to have the system requirements specification in order to follow it. As you may remember, the results of the problem domain analysis from both black- and white-box perspectives can be considered as initial system requirements specifications presented in the model-based form. The main task of this cell is to manually transform the problem domain analysis model into the SysML requirements model, which enables you to present the system requirements specification in a textual form.

After the transformation, the following traceability relationships must be established to cross the boundary between the solution and problem domain models:

- From system requirements to stakeholder needs – to assert which system requirements are derived from which stakeholder needs
- From system requirements to the rest of the problem domain model – to assert which elements are refined by which system requirement

System requirements specification can be constantly updated while working on the logical architecture of the [Sol](#). Moreover, requirements for the logical architecture of subsystems, components, and even more precise items are derived from the system requirements. This is described in Chapter [Subsystem Requirements](#).

Unlike stakeholder needs, system requirements are verifiable: after you finish with the LSA model, satisfy relationships must be established from the elements capturing the logical system architecture to the system requirements in order to assert which of the former fulfills which of the latter. This is described in Chapter [Traceability to System Requirements](#).

Who is responsible?

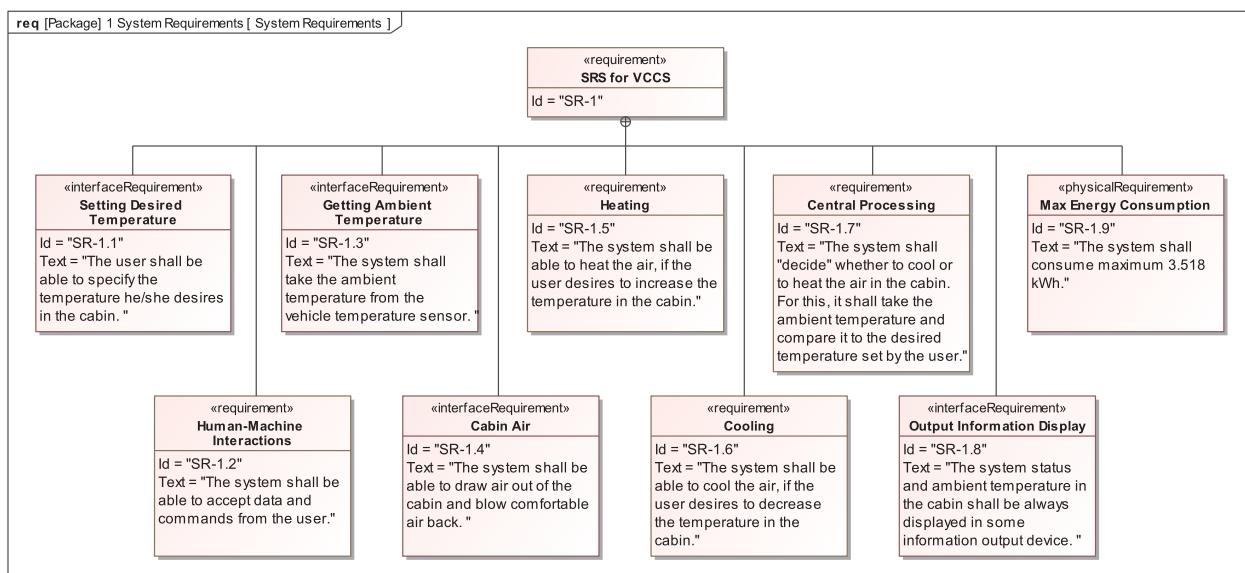
System requirements specification must be produced by the Systems Analyst or Systems Engineer who has performed the problem domain analysis.

How to model?

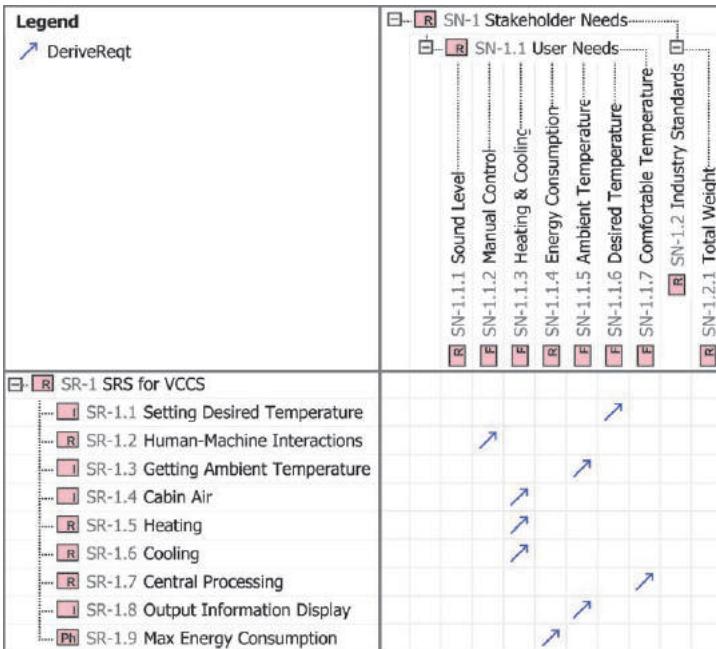
In the **modeling tool**, an item of system requirements specification can be stored as the SysML requirement, which has a unique identification, name, and textual specification. As opposed to the textual specifications of stakeholder needs, system requirements must be formal and written down following 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 the *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 (see the following figure). The table or diagram can also be included in the requirements report.



Each system requirement must be derived from a certain item of stakeholder needs and/or refine one or more elements from the problem domain model. To assert a derivation, you should establish the «deriveReqt» relationship pointing from the system requirement to the appropriate item of stakeholder needs (the following figure shows these relationships displayed in the matrix). To assert a refinement, the «refine» relationship pointing from the system requirement to some part property, call behavior action, or MoE defined in the problem domain model, should be established.

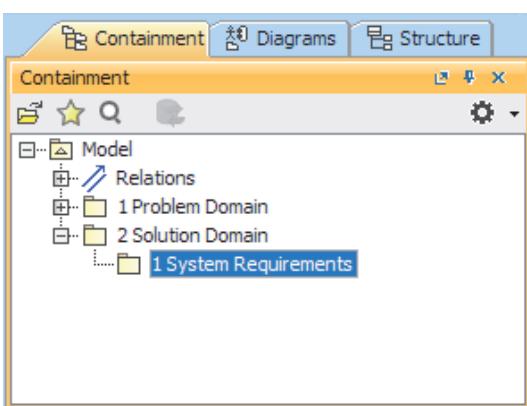


Tutorial

- Step 1. Creating and organizing a model for system requirements
- 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

Step 1. Creating and organizing a model for system requirements

To get ready for specifying system requirements, you need to establish an appropriate structure of the model first. Following the design of the MagicGrid framework, model artifacts that capture system requirements should be stored under the structure of packages, as 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 system requirements and the logical architecture of the Vehicle Climate Control System (VCCS) in another model. The contents of the problem domain model should be available in that model, because the result of the problem domain analysis is considered as an initial system requirements specification that should be transformed from the

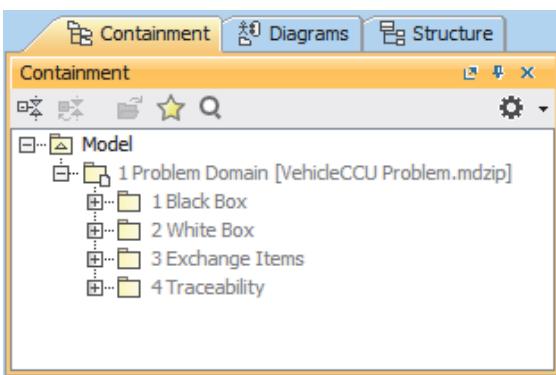
model-based to the textual form. Speaking in terms of the **modeling tool**, the problem domain model should be used in the solution domain model. For this, we utilize the relevant capabilities of the modeling tool.

To create and organize the model for system requirements specification

1. Start sharing the problem domain model.

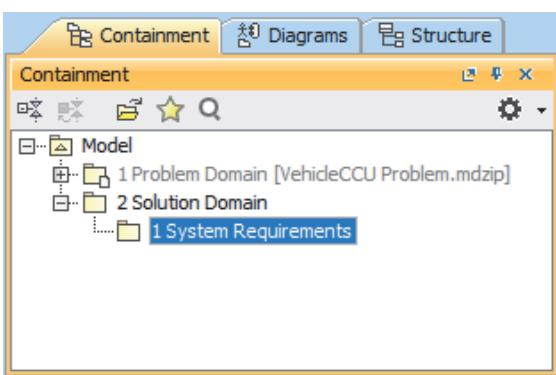
i For more information on this topic and the topics mentioned in steps 2 and 3, refer to the latest documentation of your **modeling tool**.

2. Create a new model. 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.



i 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.



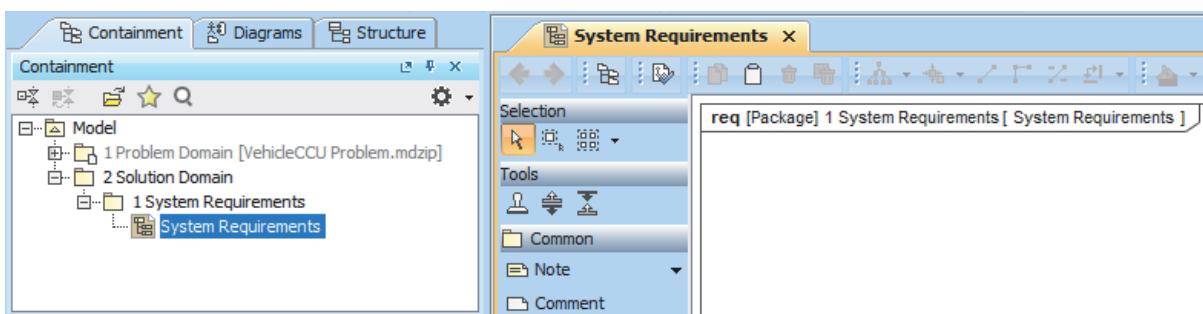
Step 2. Creating a diagram for system requirements

For capturing and displaying system requirements, we recommend using the SysML requirement diagram or table. While using the table is introduced in Chapter [Stakeholder Needs](#), using the diagram is described below.

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 double-press Enter. The diagram is created.

(i) 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

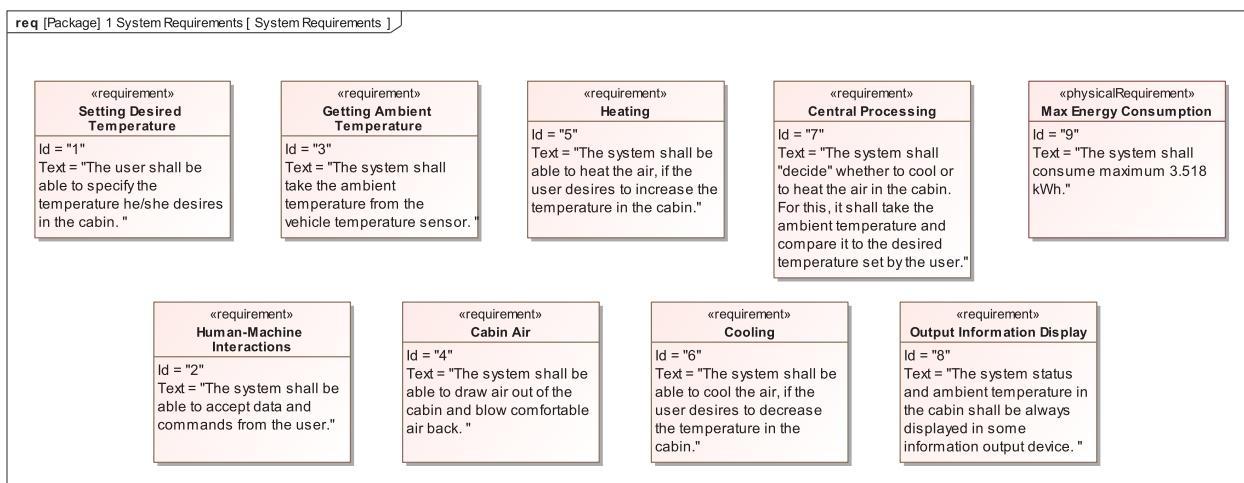
The [white-box analysis](#) of the Vehicle Climate Control Unit reveals the following system requirements:

#	Name	Text
1	Setting Desired Temperature	The user shall be able to specify the temperature he/she desires in the cabin.
2	Human-Machine Interactions	The system shall be able to accept data and commands from the user.
3	Getting Ambient Temperature	The system shall take the ambient temperature from the vehicle temperature sensor.
4	Cabin Air	The system shall be able to draw air out of the cabin and blow comfortable air back.
5	Heating	The system shall be able to heat the air, if the user wishes to increase the temperature in the cabin.
6	Cooling	The system shall be able to cool the air, if the user desires to decrease the temperature in the cabin.
7	Central Processing	The system shall "decide" whether to cool or to heat the air in the cabin. For this, it shall take the ambient temperature and compare it to the desired temperature set by the user.
8	Output Information Display	The system status and ambient temperature in the cabin shall be always displayed in some information output device.
9	Max Energy Consumption	The system shall consume maximum 3.518 kWh.

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.

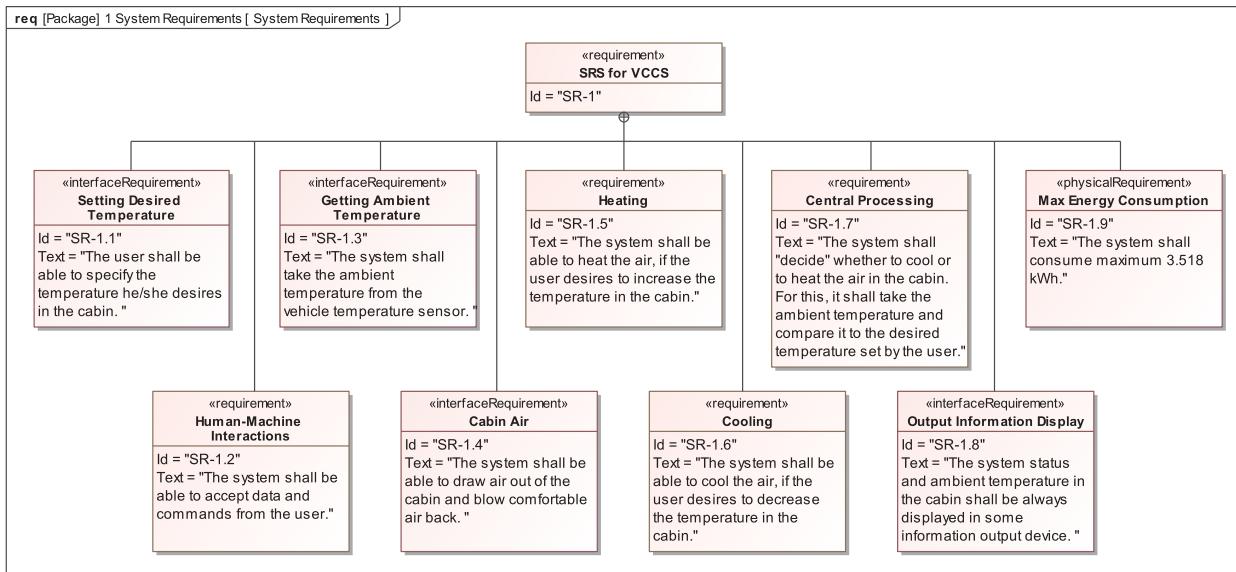


In addition, you need a grouping requirement to represent the entire requirements specification. The easiest way of creating a grouping requirement is described in [step 4](#) of Chapter [Stakeholder Needs](#) tutorial. Follow the procedure to create the *SRS for VCCS* grouping requirement.

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

1. Drag the *SRS for VCCS* requirement 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, subsystem and component requirements, and detailed requirements for the system implementation, we recommend adding prefixes to their numbers. Follow the procedure described in [step 5](#) of Chapter [Stakeholder Needs](#) tutorial to add the prefix *SR-* to your system requirements. Subsequently, these system requirements can be refactored to more specific ones. For example, *Setting Desired Temperature* can become an interface requirement and *Max Energy Consumption* a physical requirement. For this, follow the procedure described in [step 6](#) of Chapter [Stakeholder Needs](#) tutorial. After you finish 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 matrix. We recommend using the 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 concrete) are derived from what stakeholder needs (less concrete), that is, establish derivation relationships between them. Therefore, we recommend utilizing the infrastructure of the Derive Requirement Matrix, one of the predefined matrices in the [modeling tool](#). As opposed to a common matrix, it has certain criteria predefined to speed up building the view.

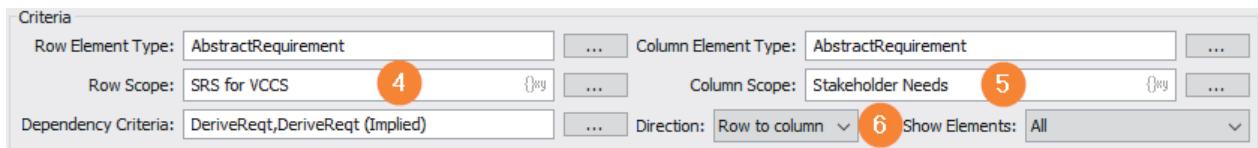
Just like in the problem domain model, traceability views and relationships can be stored in a separate package. As you already know how to create packages, we assume that you do this on your own. For keeping your model well-organized according to the MagicGrid framework, that package should always be the last sub-package of the *2 Solution Domain* package. Therefore, its name must begin with number 5, for example, *5 Traceability*.

To create a Derive Requirement Matrix

1. In the Model Browser, right-click the *5 Traceability* package and select **Create Diagram**.
 2. In the search box, type *drm* (the acronym for the predefined Derive Requirement Matrix), and press Enter.
- i** If you don't see any results, 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 *SR-1 SRS for VCCS* requirement and drag it onto the **Row Scope** box in the **Criteria** area above the matrix contents. The *SR-1 SRS for VCCS* requirement becomes the scope of the matrix rows.

5. In the Model Browser, select the *SN-1 Stakeholder Needs* requirement (you might need to expand the *1 Problem Domain, 1 Black Box, and 1 Stakeholder Needs* packages first) and drag it onto the **Column Scope** box in the **Criteria** area above the matrix contents. The *SN-1 Stakeholder Needs* requirement 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 (row elements), to stakeholder needs (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.

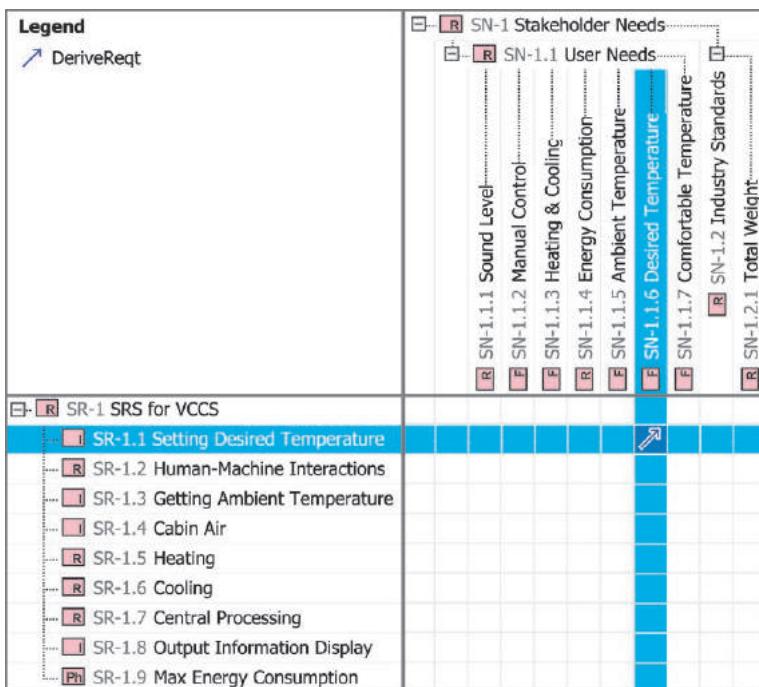
The following figure displays the **Criteria** area of the matrix, with highlights on step 4, 5, and 6 related criteria.



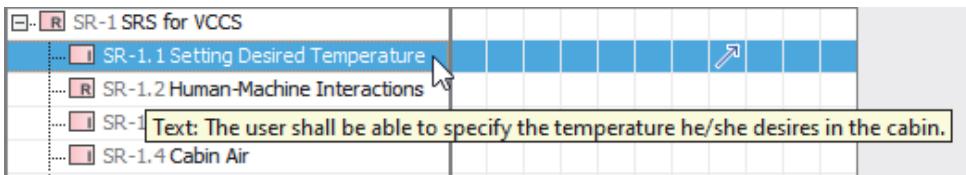
Now you're ready to establish derivation relationships. Let's create one between *SR-1.1 Setting Desired Temperature* and *SN-1.6 Desired Temperature* to convey that *SR-1.1 Setting Desired Temperature* is derived from *SN-1.6 Desired Temperature* or, in other words, that this system requirement is created because of that stakeholder need.

To specify the derivation relationship in the matrix

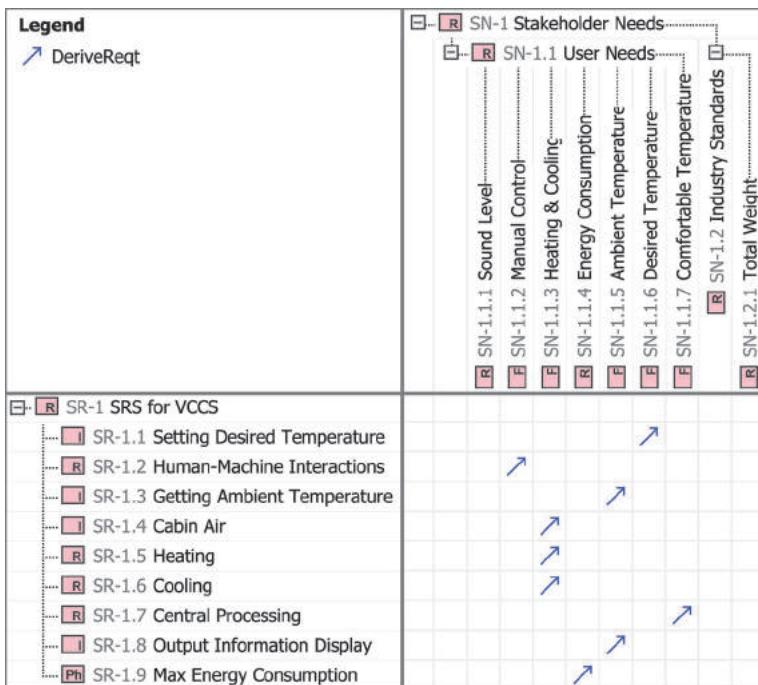
- Double-click the cell at the intersection of the row that displays *SR-1.1 Setting Desired Temperature* and the column that displays *SN-1.6 Desired Temperature*. The derivation relationship is created between the appropriate items and displayed in the cell.



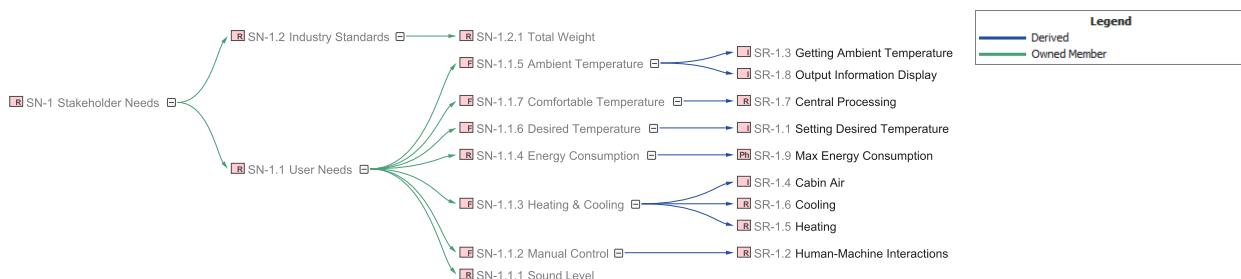
- i** If you want to read the requirement text before setting the relationship, just move the mouse over that requirement name and rest it there for a few seconds. The text appears in the tool tip.



After all derivation relationships are established, your *System Requirements to Stakeholder Needs* matrix should resemble the one in the following figure. You may remember that every system requirement must be derived from one or more stakeholder needs. Otherwise, the system requirements specification must be revised and updated. Some stakeholder needs may be left uncovered, if it was decided not to address them in the system requirements specification.



Afterwards, a Requirement Derivation Map can be created, with blue arrows representing the relationships between requirements from different domains.



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 matrix. We recommend using the 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 which problem domain elements are refined by which system requirements; that is, establish refine relationships between them. Therefore, we recommend utilizing the infrastructure of the Refine Requirement Matrix, one of the predefined matrices in the modeling tool. As opposed to a common matrix, it has certain criteria predefined to speed up building the view.

This matrix and appropriate relationships can be stored in the *5 Traceability* package you created in the previous step.

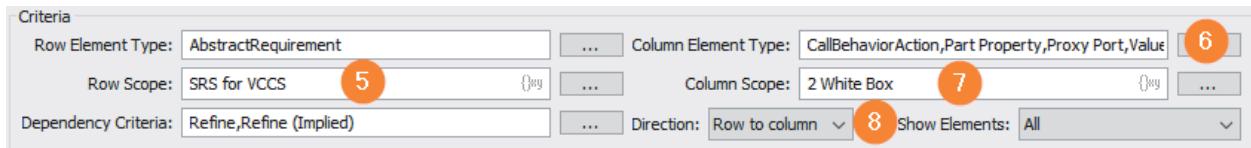
To create a Refine Requirement Matrix

1. In the Model Browser, right-click the *5 Traceability* 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 results in the diagram types list, click the **Expert** button below it. 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 *SR-1 SRS for VCCS* requirement (you might need to expand the *1 System Requirements* package first) and drag it onto the **Row Scope** box in the **Criteria** area above the matrix contents. The *SR-1 SRS for VCCS* requirement becomes the scope of the matrix rows.
6. Select column element types:
 - a. In the Model Browser, select any call behavior action; for example, the one typed by the *Initialize Cooling* activity (it appears under the *Reach Desired Temperature* activity which can be found within the *3 Functional Analysis* package in the problem domain analysis model).
 - b. Drag the selection onto the **Column Element Type** box in the **Criteria** area above the matrix contents. The columns of the matrix are set to display the elements of the call behavior action type.
 - c. In the Model Browser, select any part property, such as the one typed by the *Cooling* block. It appears under the *VCCU* block, which can be found within the *2 Logical Architecture* package in the problem domain analysis model.
 - d. Hold down the Ctrl key and drag the selection onto the **Column Element Type** box in the **Criteria** area above the matrix contents. The columns of the matrix are set to display the elements of the part property type as well.
 - e. Repeat steps c and d to enable the columns of the matrix to display the proxy ports and value properties with the «moe» stereotype.
7. Specify the column element scope:
 - a. In the Model Browser, select the *2 White Box* package in the problem domain analysis model.
 - b. Drag the selection onto the **Column Scope** box in the **Criteria** area above the matrix contents. The selected package becomes the scope of the matrix columns.

- Click the **Direction** box and select **Row to Column**, as you need to establish refine relationships pointing from system requirements (displayed as row elements), to the artifacts that capture problem domain concepts (displayed as column elements).
- Click the **Refresh** hyperlink in the notification box below the **Criteria** area. The contents of the matrix are updated. All cells are empty at the moment.

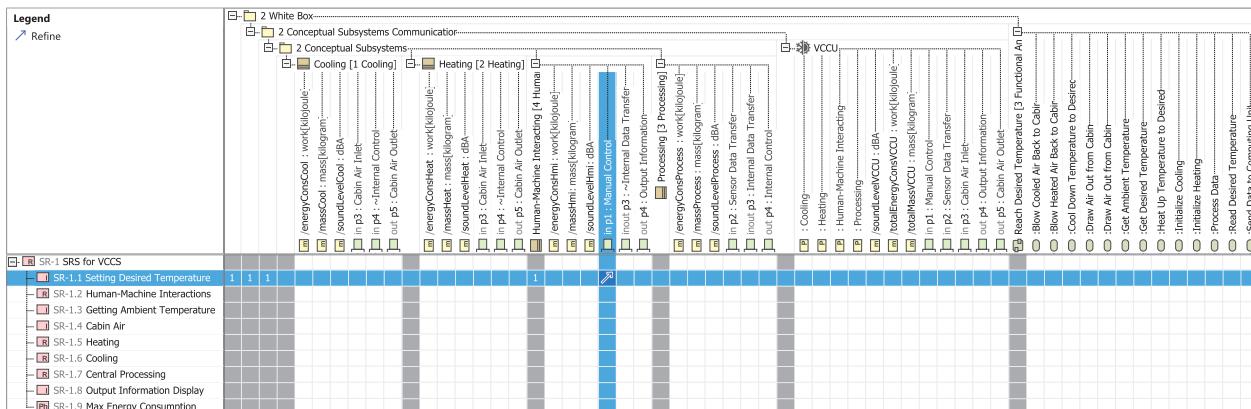
The following figure displays the **Criteria** area of the matrix with highlights on steps 5, 6, 7, and 8 related criteria.



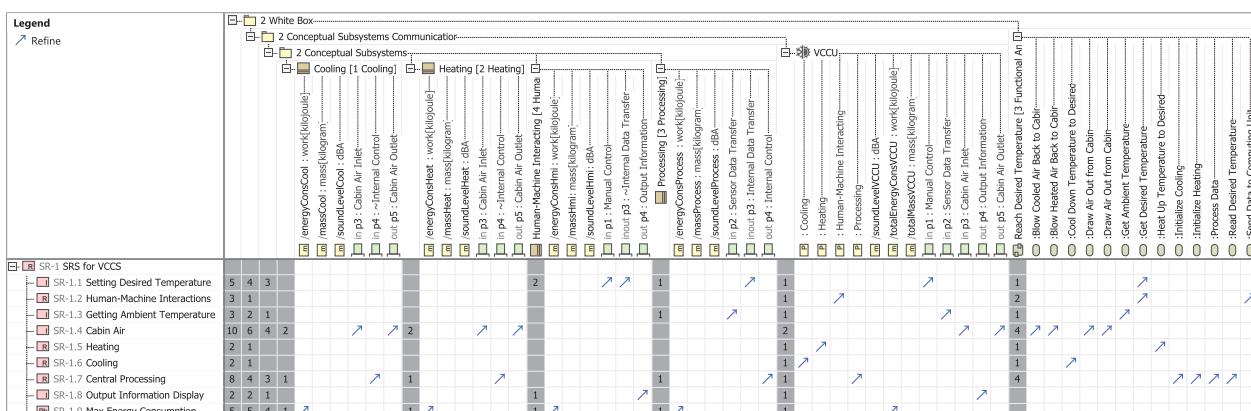
Now you're ready to establish refine relationships. Let's create one between the *SR-1.1 Setting Desired Temperature* requirement and the *p1* proxy port of the *Human-Machine Interacting* block to assert that the former refines the latter, or, in other words, that this system requirement is created to refine that proxy port.

To specify the refine relationship in the matrix

- Double-click the cell at the intersection of the row that displays *SR-1.1 Setting Desired Temperature* and the column that displays the *p1* proxy port. The refine relationship is created between appropriate items and displayed in the cell.



After all refine relationships are specified, your *System Requirements to Problem Domain* matrix should resemble the one in the following figure. Keep in mind that every system requirement must refine one or more elements captured in the problem domain analysis model. Otherwise, the system requirements specification must be revised and updated. Some elements can be left uncovered, if it was decided not to address them in system requirements specification (such as MoEs for the sound level and total weight).



System Requirements done. What's next?

- When you have the system requirements specification captured in the SysML requirements model, you can start building the logical architecture of the **Sol**. This means you can move to the **System Structure** cell.
- System requirements specification can be updated any time while building the logical architecture of the **Sol**. The updates should stop only after switching to the subsystem-level solution domain model.

System Structure

What is it?

After the system requirements specification is completed, you can start building the logical architecture of the **Sol**. Usually it starts with capturing the logical subsystems of the **Sol**. The LSA model also specifies interfaces, to determine how these subsystems shall 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 conceptual architecture in the problem domain model, cross-cutting relationships can be established.

Defining the logical subsystems serves for the purpose of establishing the work-breakdown structure (WBS) in the systems engineering project: each logical subsystem identifies a single work package, which is immediately appointed to some engineering team. That team develops one or more logical architectures for the appointed subsystem. Separate teams usually work in parallel to develop logical architectures of different subsystems. Modeling the logical architecture of a single subsystem is described in Chapter [Building the logical architecture of subsystems](#).

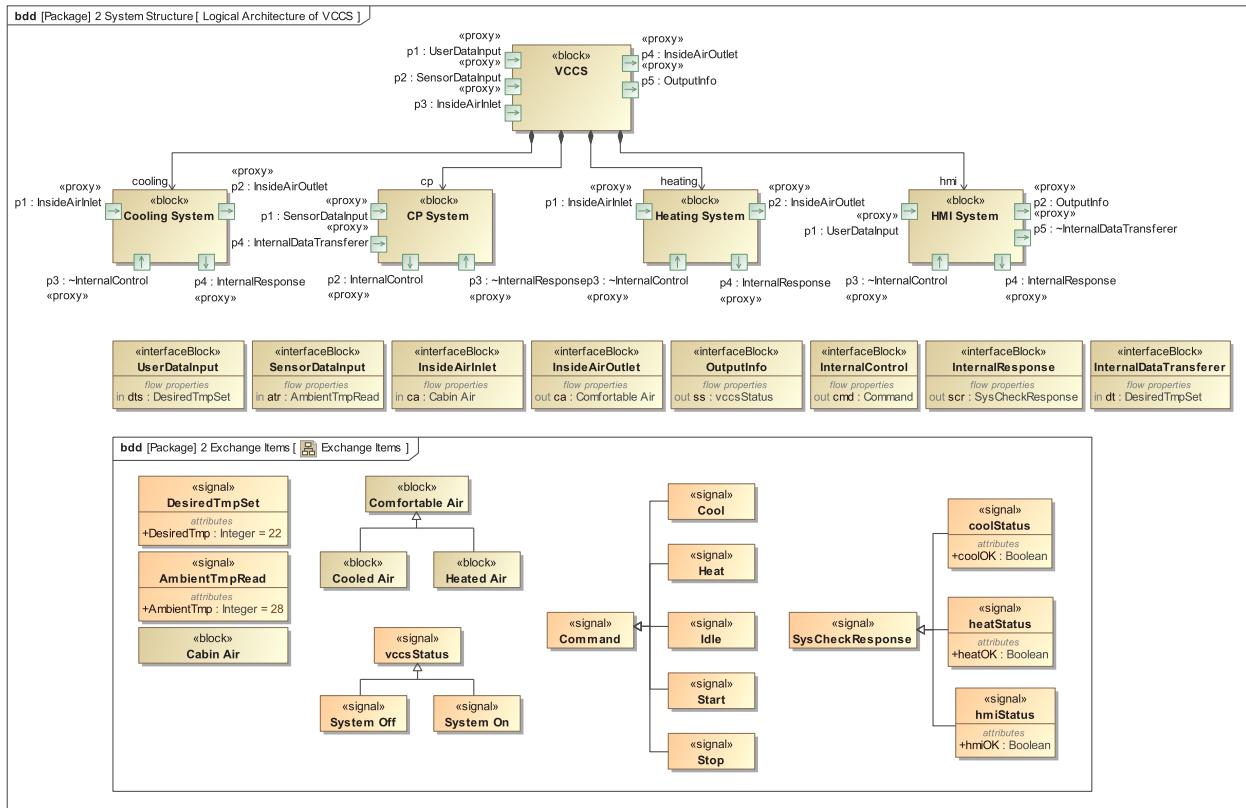
Who is responsible?

System structure in the LSA model can be captured by the Systems Architect or Systems Engineer. He/she must be the head of the whole systems engineering project, the one, who has "a big picture view" of the **Sol** and is responsible for the smooth integration of the subsystems and their components into the whole.

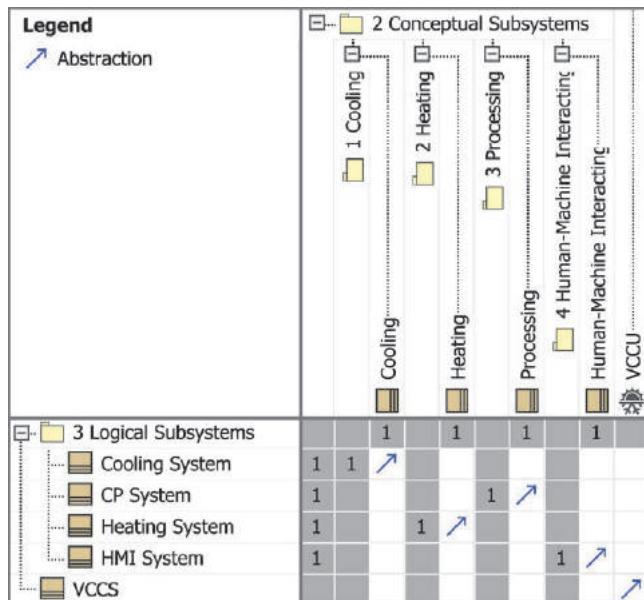
How to model?

As you can remember, the LSA captures the **system of interest**, its logical subsystems at one level down, and the interfaces to determine how these subsystems inter-operate, that is, what data, matter, or energy they exchange. The LSA model can be created and displayed in a SysML bdd. Subsystems can be captured as blocks, and interfaces as proxy ports typed by appropriate interface blocks.

In the following figure, you can see a sample LSA model of the Vehicle Climate Control System (VCCS). It displays the blocks that capture logical subsystems of the VCCS, interface blocks that own flow properties which capture inputs and outputs for both external and internal communications as well as various exchange items which type relevant flow properties.



The «abstraction» relationship can be used to declare what logical subsystems in the LSA are derived from what conceptual subsystems (functional blocks) in the problem domain analysis model.

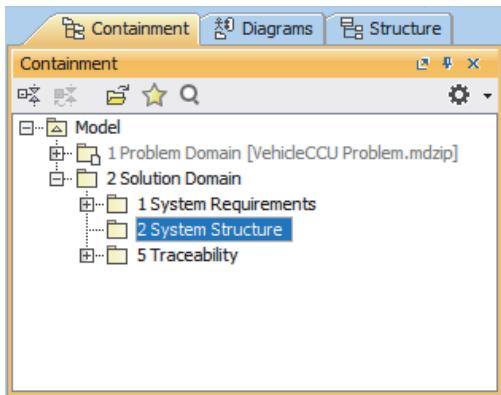


Tutorial

- Step 1. Organizing the model for the LSA
- Step 2. Creating a bdd for capturing the LSA
- Step 3. Capturing logical subsystems
- Step 4. Establishing traceability to the problem domain model
- Step 5. Capturing logical interfaces

Step 1. Organizing the model for the LSA

The LSA can be captured in the solution domain model you created in Chapter [System Requirements](#). To get ready for modeling in this cell, you need to establish an appropriate structure of the model beforehand. Following the design of the MagicGrid framework, relevant model artifacts should be stored under the structure of packages, as displayed in the following figure.



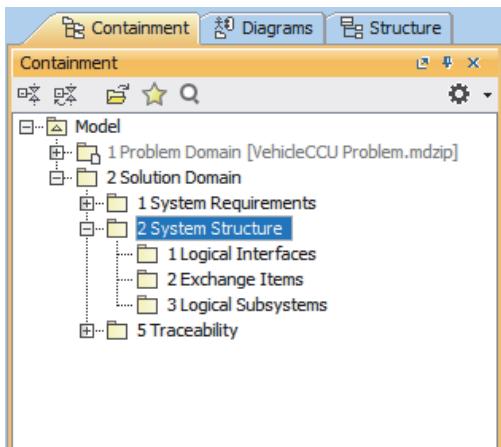
To organize the model for the LSA

1. Open the solution domain model (the *VehicleCCS Solution.mdzip* file) you created in Chapter [System Requirements](#), 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 *2 System Structure* to specify the name of the new package and press Enter.

For the purpose of keeping the inner structure of the *2 System Structure* package well organized, you need to create a few more packages. These are:

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

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



Step 2. Creating a bdd for capturing the LSA

To get ready for capturing the logical architecture of the VCCS, you need to create a bdd first.

To create a bdd for capturing the logical architecture of the VCCS

1. Right-click the *2 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 *Logical Architecture of VCCS* to specify the name of the new diagram and press Enter again.

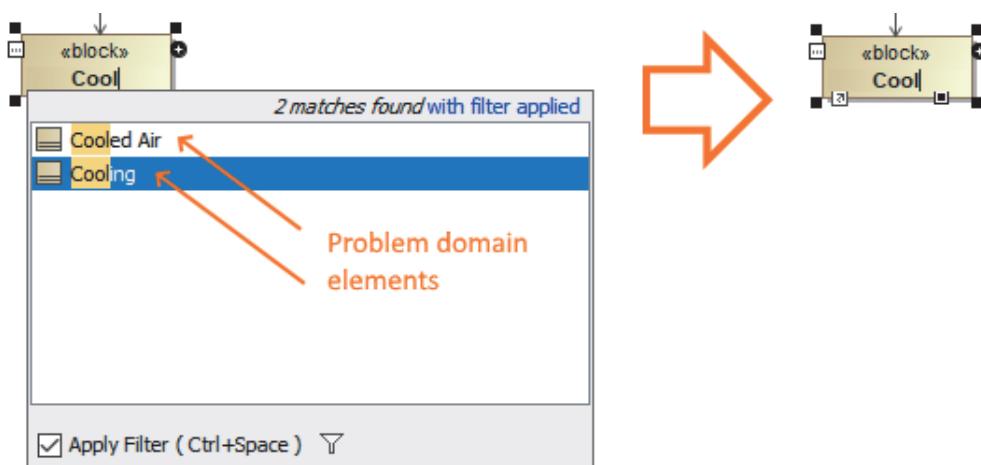
Step 3. Capturing logical subsystems

As the system requirements specification reveals, the VCCS consists of:

- HMI System, where *HMI* stands for *Human-Machine Interface*
- CP System, where *CP* stands for *Central Processing*
- Cooling System
- Heating System

Logical subsystems can be captured as blocks in the **bdd** you created in the previous step. As already mentioned, the LSA model always defines logical subsystems at one level down, and the logical architecture of each subsystem should be subsequently produced by separate engineering teams in isolated models (see Chapter [Subsystem Structure](#)).

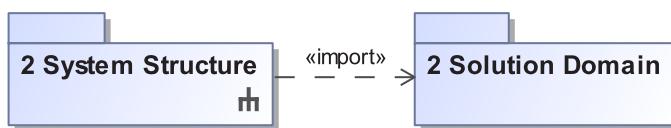
Note that the names of the logical subsystems correspond to the names of the conceptual subsystems in the problem domain model, such as Cooling System to Cooling, or Heating System to Heating. However, these concepts are not the same; therefore, be sure you haven't selected the block from the problem domain model instead of creating a new one in the solution domain model, when specifying the logical subsystems of the VCCS. To prevent this mistake, you can narrow down the name auto-completion and type selection scope beforehand. The following figure displays the auto-completion before (on the left) and after (on the right), narrowing the scope to the solution domain model only.



To narrow the auto-completion and type selection scope down to the solution domain model

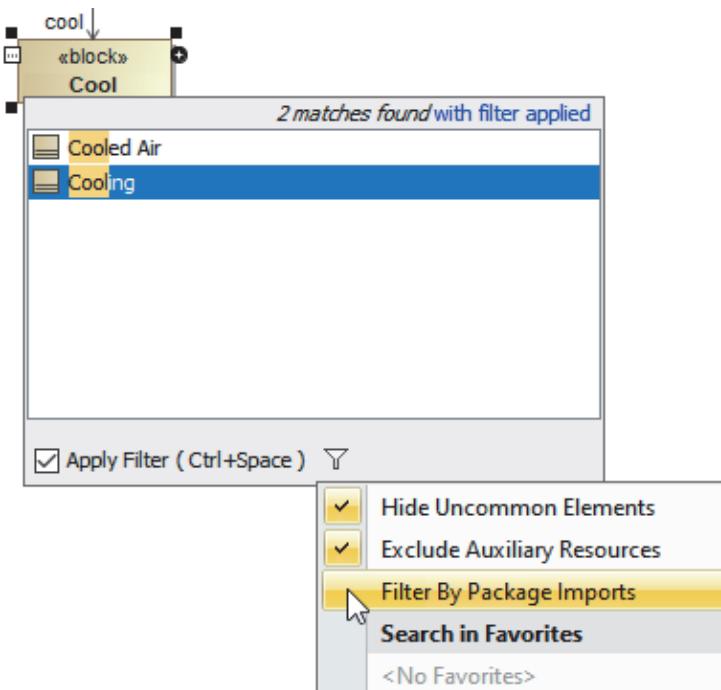
1. Create a package diagram:

- a. Right-click the *2 System Structure* package and select **Create Diagram**.
 - b. In the search box, type *pd*, where *p* stands for *package* and *d* for *diagram*, and then press Enter. The diagram is created.
 - c. Type *Scope for Auto-completion and Type Selection* to specify the name of the new diagram and press Enter again.
2. In the Model Browser, select the *2 Solution Domain* and *2 System Structure* packages (the latter is a sub-package of the former) and drag them to the pane of the newly created diagram.
3. Select the shape of the *2 System Structure* package and click the Package Import button  on its smart manipulator toolbar.
4. Select the shape of the *2 Solution Domain* package. The package import relationship is created between the packages.



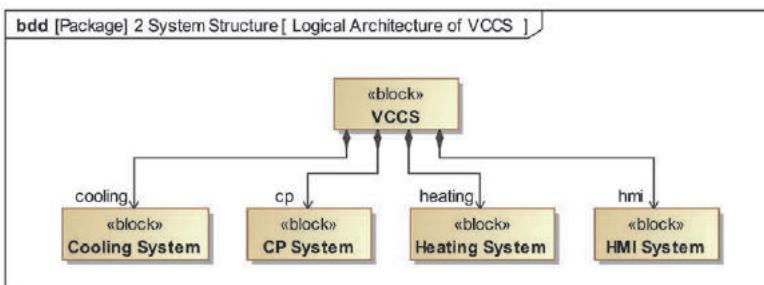
From now on, when you model in the *2 System Structure* package, elements suggested as names and types will be taken from the *2 Solution Domain* package only, while elements stored in the *1 Problem Domain* package (from the problem domain model) will be left aside. This means it won't be suggested that you choose from conceptual subsystems and interfaces as well as exchange items captured in the problem domain model.

If you still see those elements, click the Filter Options button  below the list and then click to select the **Filter By Package Imports** check box (see the following figure). For more information about the package import feature, refer to the latest documentation of the [modeling tool](#).

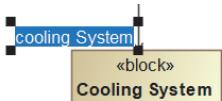


To capture logical subsystems of the **Sol**

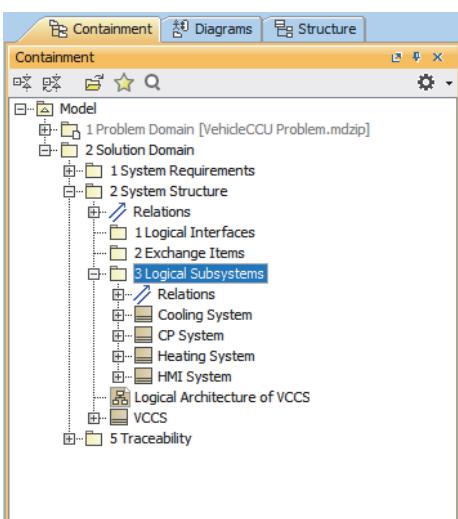
1. Open the *Logical Architecture of VCCS* diagram, if not opened yet.
2. Capture the **Sol** in the solution domain model:
 - a. Click the **Block** button on the diagram palette and then click on the diagram pane.
 - b. Type *VCCS* to name the block and press Enter.
3. Capture the logical subsystems:
 - a. Click the Directed Composition button  on the smart manipulator toolbar of the *VCCS* block and then click an empty place on the diagram pane.
 - b. Type *HMI 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 .



i Note that the part property names (displayed at the arrow end of the relationship) differ from the default. They can be changed manually. For this, you need to select the part property you want to rename on the diagram and then click it again to enter the name edit mode (see the following figure). You are able to modify it immediately.



5. In the Model Browser, select all blocks of the subsystems and drag them to the *3 Logical Subsystems* package you created in [step 1 of this cell tutorial](#). The blocks are moved to the *3 Logical Subsystems* package.



Step 4. Establishing traceability to the problem domain model

Traceability relationships enable you to specify which conceptual subsystems (aka functional blocks) identified in the problem domain model determine the creation of what logical subsystems in the LSA model. We use the abstraction relationships to trace from the solution domain to the problem domain in this case.

You already know that the most effective view for establishing traceability relationships is a matrix. Let's create one.

To create the matrix for traceability relationships to the problem domain model

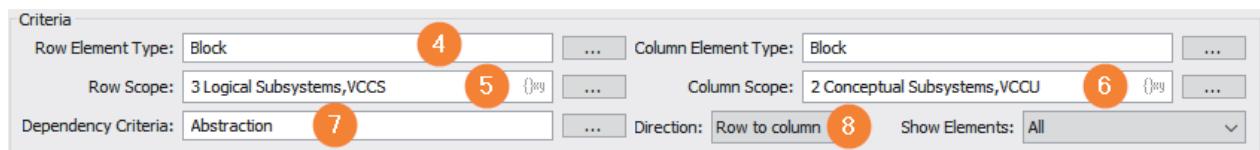
1. In the Model Browser, right-click the *5 Traceability* package, a subpackage of the *2 Solution Domain* package, and select **Create Diagram**.
2. In the search box, type *dm* (the acronym for the Dependency Matrix), and press Enter.

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

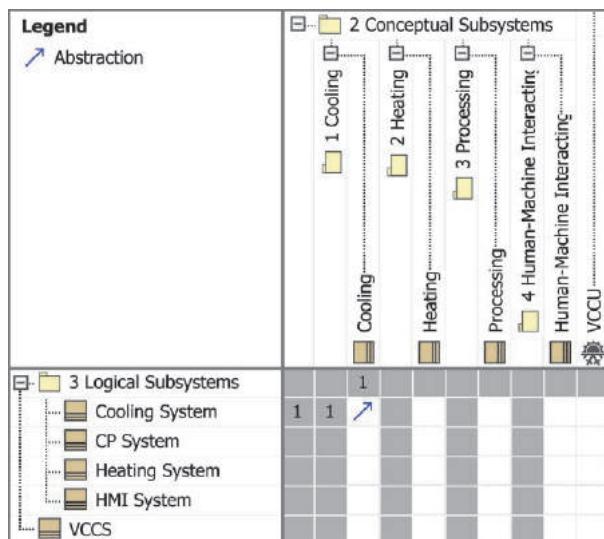
3. Type *LSA to Problem Domain* to specify the name of the new matrix and press Enter again.
4. Specify the row and column element type:
 - a. In the Model Browser, select any block, such as the *VCCS* block.
 - b. Drag it onto the **Row Element Type** box in the **Criteria** area above the matrix contents. Rows and columns of the matrix are both set to display the elements of the block type (see the figure below).
 - c. Click the button next to the **Column Element Type** box.
 - d. In the open dialog, click to clear the **Include Subtypes** check-box. Otherwise, the columns display a few interface blocks, which you don't need here.
 - e. Click **OK** to close the dialog.
5. Specify the row element scope:
 - a. In the Model Browser, select the *VCCS* block.
 - b. Drag the selection onto the **Row Scope** box in the **Criteria** area above the matrix contents. The selected block becomes the scope of the matrix rows.
 - c. In the Model Browser, select the *3 Logical Subsystems* package, which appears under the same package with the *VCCS* block.
 - d. Hold down the Ctrl key and drag the selection onto the **Row Scope** box, too. The *3 Logical Subsystems* package becomes the scope of the matrix rows, together with the *VCCS* block (see the following figure).
6. Specify the column element scope:
 - a. In the Model Browser, select the *VCCU* block (for this, you might need to expand the contents of the following read-only packages: *1 Problem Domain > 2 White Box > 2 Conceptual Subsystems Communication*).
 - b. Drag the selection onto the **Column Scope** box in the **Criteria** area above the matrix contents. The selected block becomes the scope of the matrix columns.
 - c. In the Model Browser, select the *2 Conceptual Subsystems* package, which appears under the same package with the *VCCU* block.

- d. Hold down the Ctrl key and drag the selection onto the **Column Scope** box, too. The *2 Conceptual Subsystems* package becomes the scope of the matrix columns, together with the *VCCU* block (see the following figure).
7. Specify the dependency criteria:
- Click the button next to the **Dependency Criteria** box.
 - In the open dialog, find the abstraction relationship. For this:
 - Type *abs* in the search box below the criteria list.
 - Click to select the check box next to the abstraction relationship.
 - Click **OK**. The abstraction relationship becomes the dependency criteria of the matrix (see the following figure).
8. Click the **Direction** box and select **Row to Column**, as you need to establish the abstraction relationships pointing from logical subsystem (row elements), to conceptual subsystems (column elements).
9. 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.

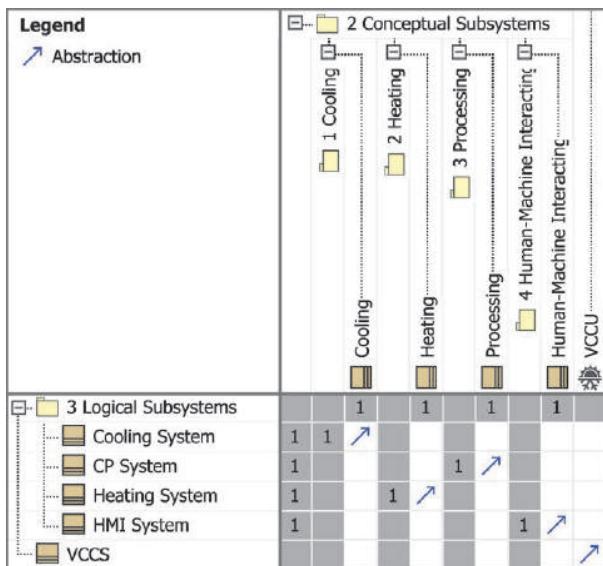
The following figure displays the **Criteria** area of the matrix with highlights on step 4, 5, 6, 7, and 8 related criteria.



Now you're ready to specify abstraction relationships. Double-click the cell at the intersection of relevant row and column (for example, between the row that represents the *Cooling System* block and the column that represents the *Cooling* block).



After all abstraction relationships are specified, the *LSA to Problem Domain* matrix in your model should resemble the one in the following figure.

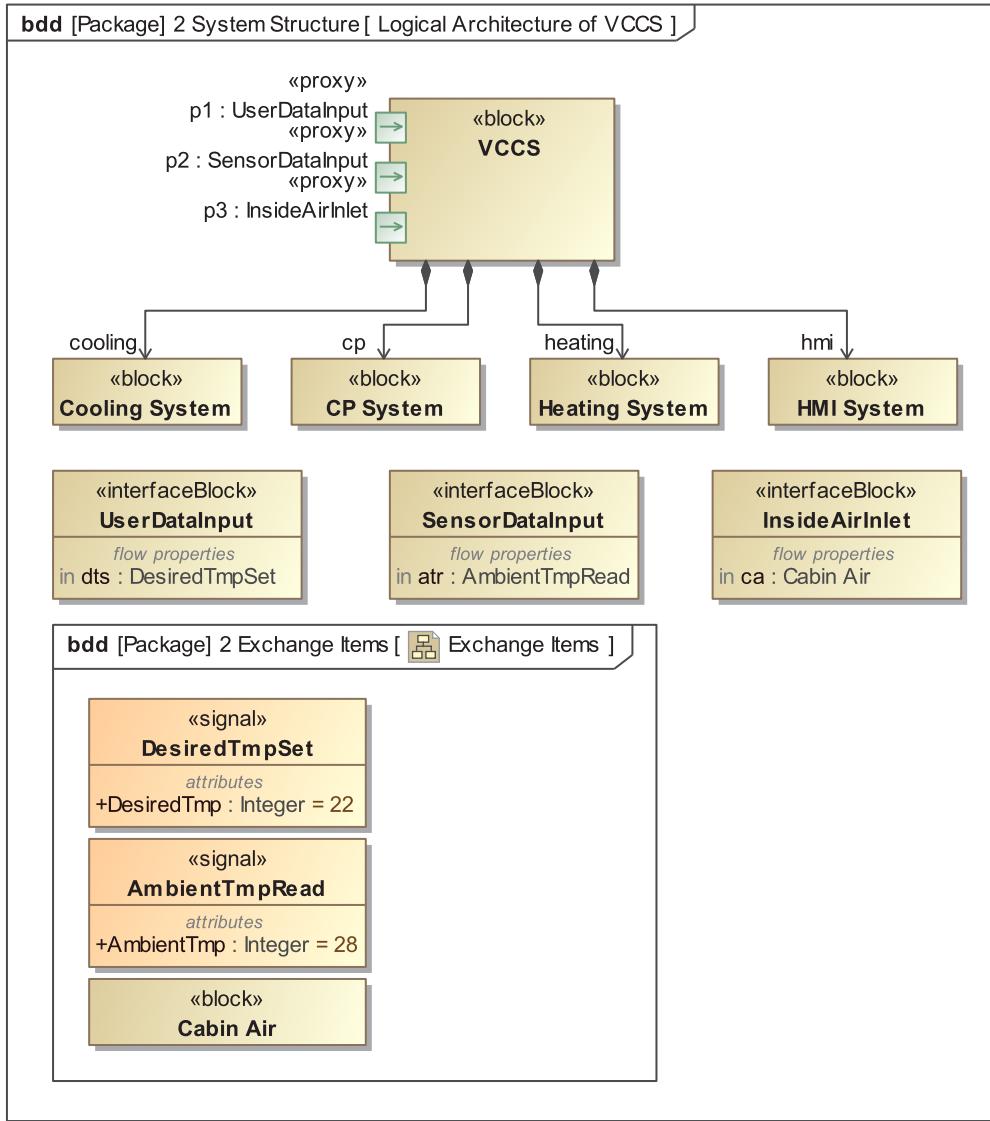


Step 5. Capturing logical interfaces

As the system requirements specification reveals, the VCCS shall be able to receive the following from the outside:

- Desired temperature specified by the user
- Ambient temperature measured by the temperature sensor of the vehicle
- Cabin air for cooling or heating

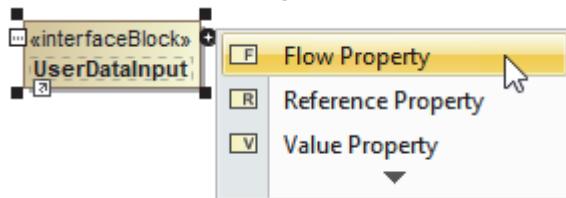
These statements indicate the need for the logical interfaces that you see in the following figure. These interfaces are more precise than conceptual interfaces determined in the problem domain analysis model: the *DesiredTmpSet* and *AmbientTmpRead* signals have properties for holding specific values of Integer type. These properties can even have default values, such as *DesiredTmp=22*.



If you've gone through the tutorial in Chapter [Conceptual Subsystems](#), you should already know how to create interface blocks with flow properties and how to bind these interface blocks to system architecture blocks. If you've skipped it, here is another chance for you to learn.

To capture the interface block for providing the desired temperature and relate it to the VCCS block

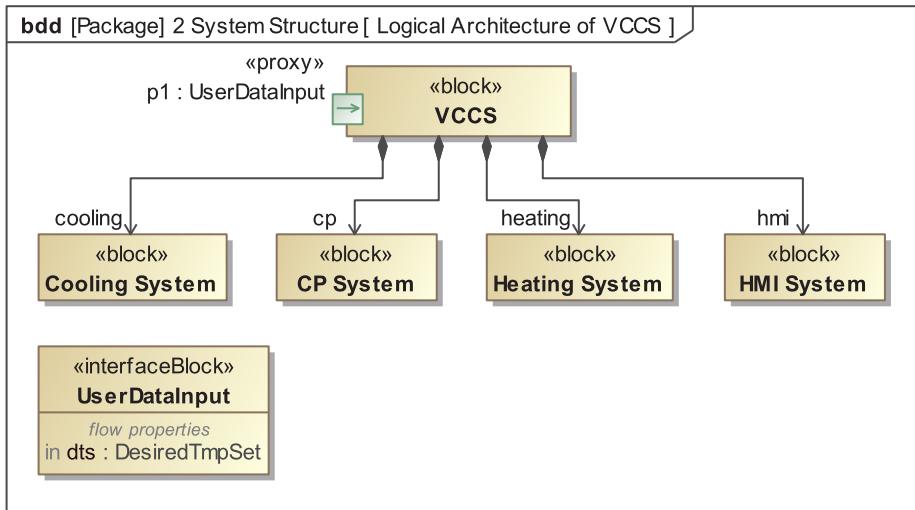
1. Open the *Logical Architecture of VCCS* diagram, if not opened yet.
2. Click the **Interface Block** button on the diagram palette and then click an empty place on the diagram pane. An unnamed interface block is created in the model, directly under the *2 System Structure* package, and displayed on the diagram.
3. Type *UserDataInput* to name the interface block and press Enter.
4. Click the Create Property button on the interface block shape and select **Flow Property**.



5. 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 *dts* to specify the flow property name. Don't press Enter!
 - c. Type *:DesiredTmpSet* and press Enter. The *DesiredTmpSet* signal is created in the model directly under the *UserDataInput* interface block and immediately set as its only flow property type.

6. Arrange your model:
 - a. Make sure the *UserDataInput* interface block is selected on the diagram and press Alt+B. The element is selected in the Model Browser.
 - b. Click  near the *UserDataInput* interface block to expand its contents.
 - c. Select the *DesiredTmpSet* signal and drag it to the *3 Exchange Items* package.
 - d. Select the *UserDataInput* interface block again and drag it to the *1 Logical Interfaces* package, a sub-package of the *2 System Structure* package.

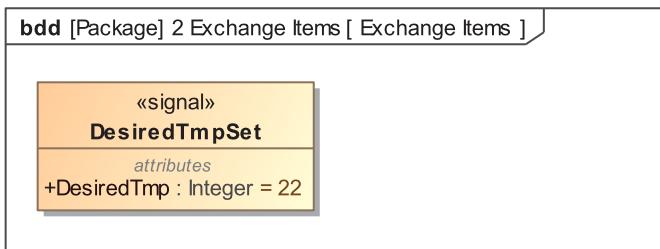
7. Select the *UserDataInput* interface block on the diagram and drag it onto the *VCCS* block. As a result, a new proxy port, typed by the *UserDataInput* interface block, is created for the *VCCS* block. Note that it is decorated with an arrow indicating that this port is for inputs to the **system of interest**.



8. Create a diagram for displaying exchange items:
 - a. In the Model Browser, right-click the *3 Exchange Items* 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. Remove the sequence number from the diagram name press Enter again.

9. Create a property for the *DesiredTmpSet* signal:
 - a. In the Model Browser, select that signal and drag it to the *Exchange Items* diagram pane.
 - b. Click the Create Property button  on the signal shape.
 - c. Type *DesiredTmp* to name the new property.
 - d. Type *:Int* press Ctrl + Spacebar to open the list of possible types. Once you see the *Integer* value type (the pink one) in the search results list, select it.

- e. Type =22 and press Enter. The property of the *Integer* type with 22 as the default value is created for the signal.

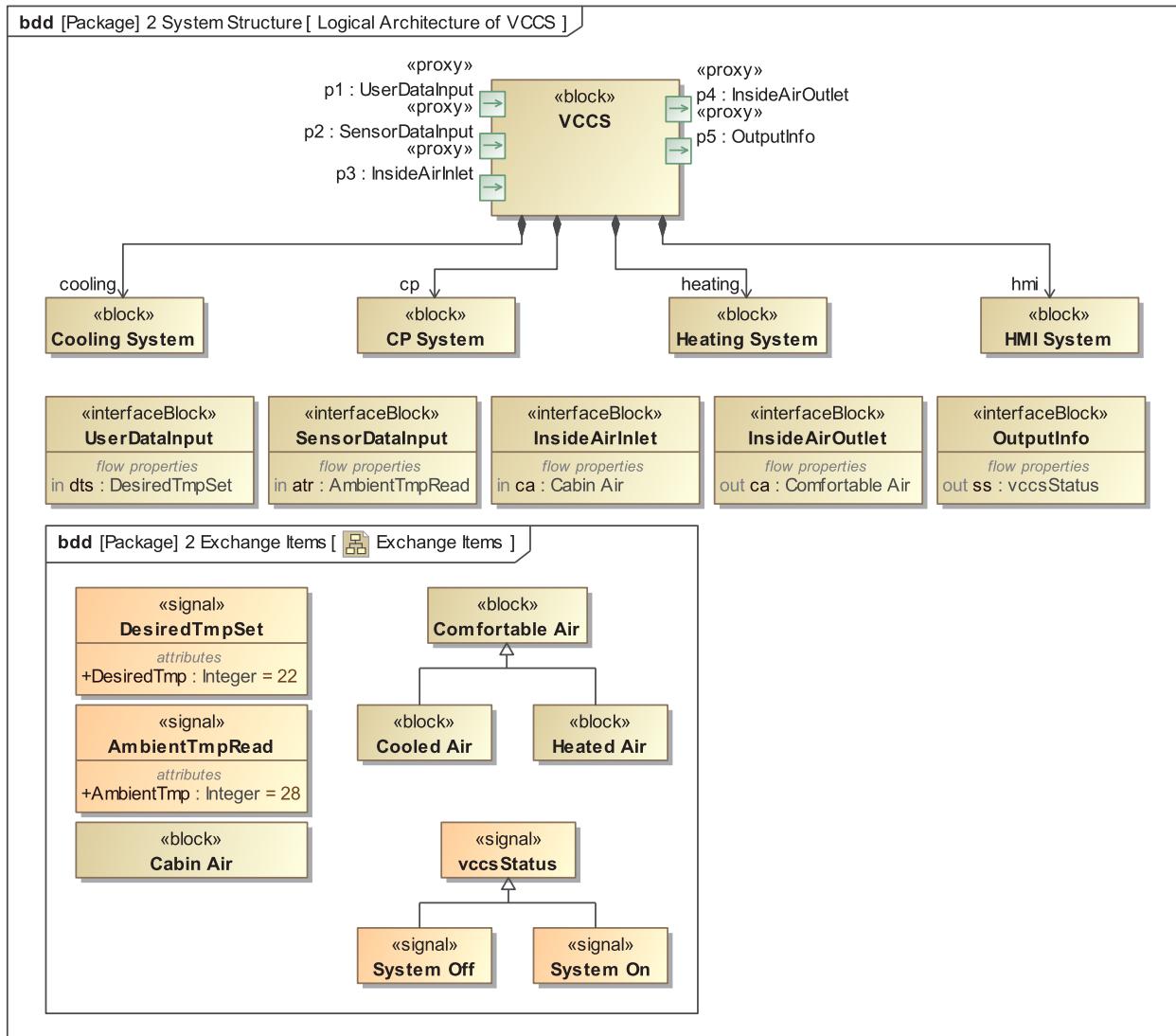


Following the previous procedure, capture another two interface blocks (with all the related information) and bind them to the *VCCS* block. Once you're done, the *Logical Architecture of VCCS* diagram in your model should look very similar to the first image of this tutorial step.

The system requirements specification also reveals that the *VCCS* shall be able to provide the following to the outside:

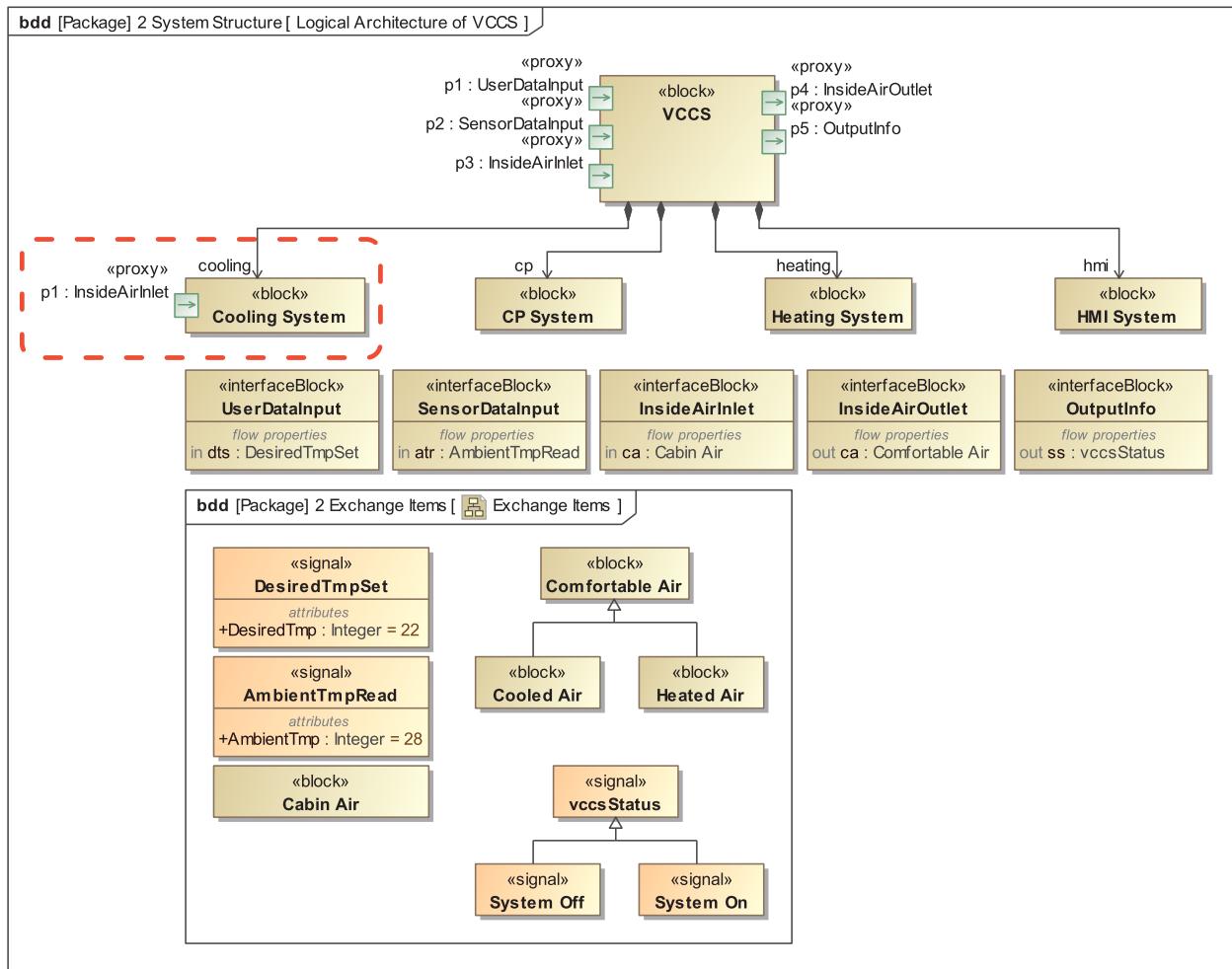
- Comfortable (cooled or heated) air back to the cabin
- Output information about the *VCCS* status for the user

These statements indicate that the *VCCS* needs a few more interfaces. You can see them in the following figure. We assume that you update the model on your own.

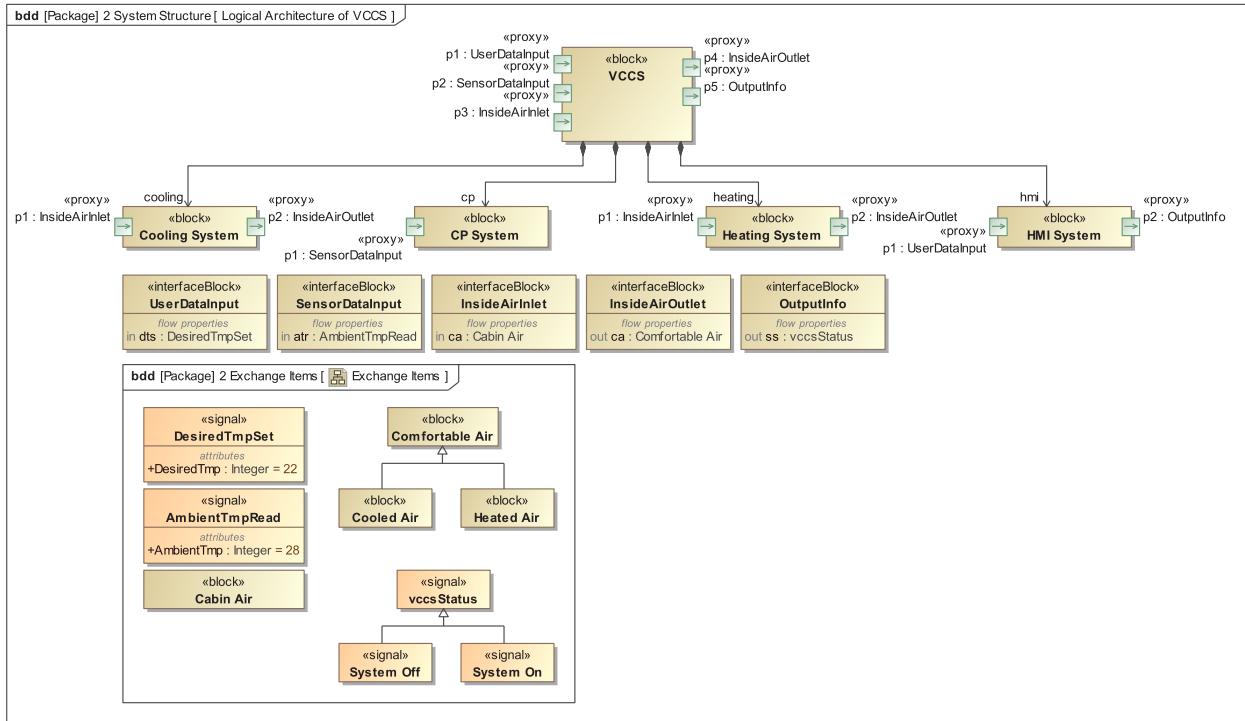


Now it's time to start thinking about the logical interfaces of logical subsystems of the VCCS. Just like the **Sol** itself, each logical subsystem can have a number of logical interfaces. Some of them can be for external communication. Therefore, they are inherited from the **Sol**. Some are for internal communication with other subsystems of the VCCS. These are specified in the LSA model from scratch.

Let's start from specifying logical interfaces for external communication. As already mentioned, you don't need to create new interface blocks: simply drag the selected interface block onto the block that captures the relevant logical subsystem (the same as you did for the *VCCS* block). For example, you can drag the *InsideAirInlet* interface block onto the *Cooling System* block. As a result, a proxy port typed by the *InsideAirInlet* interface block is created for the *Cooling System* block.

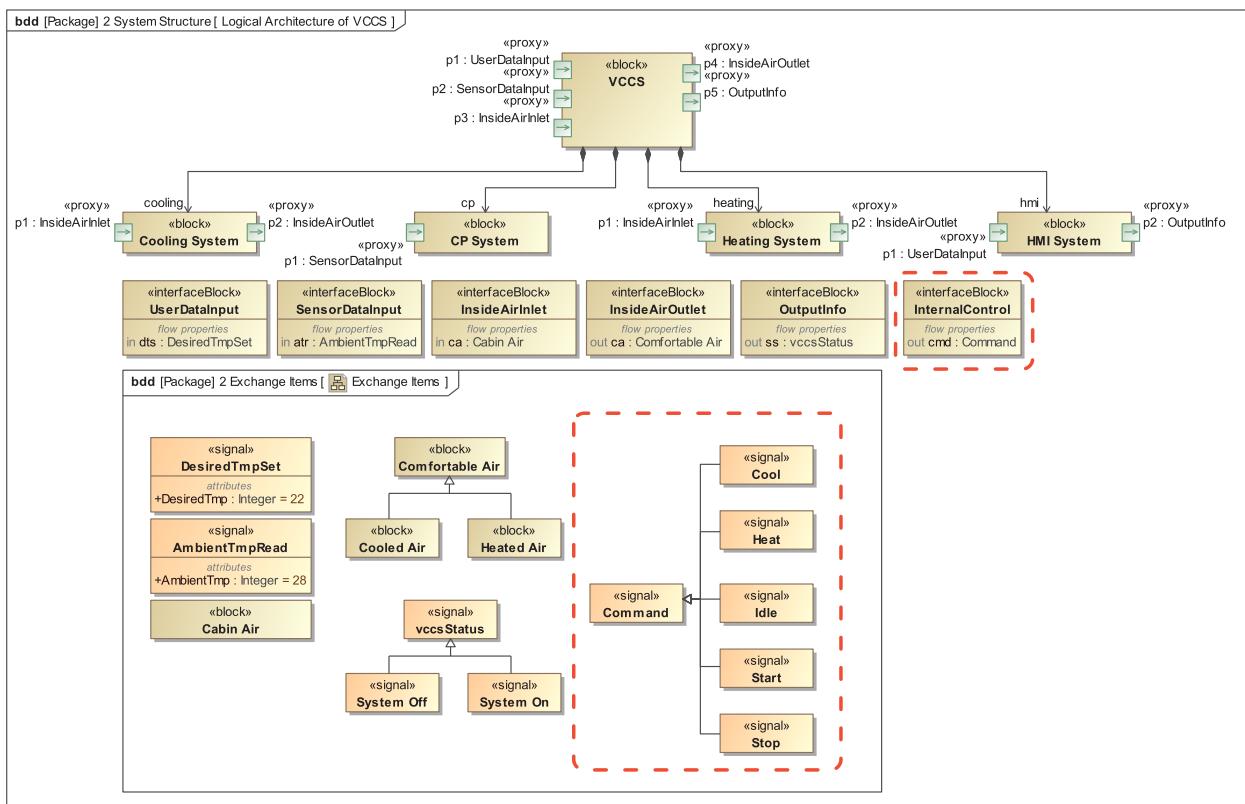


After you specify all the interfaces for external communication, the *Logical Architecture of the VCCS* bdd in your model should look very similar to the one in the following figure.



Now let's think about the logical interfaces for internal communication between logical subsystems.

It is obvious that the CP System shall be able to send commands to other subsystems. These might include Start, Stop, Idle, Cool, Heat, etc. They can be identified later, when modeling the logical architecture of that subsystem. This requires creating a new interface block and a few signals (you can see them highlighted in the following figure). Also, new subsystem requirement must added, but let's leave it for modeling in other cells.

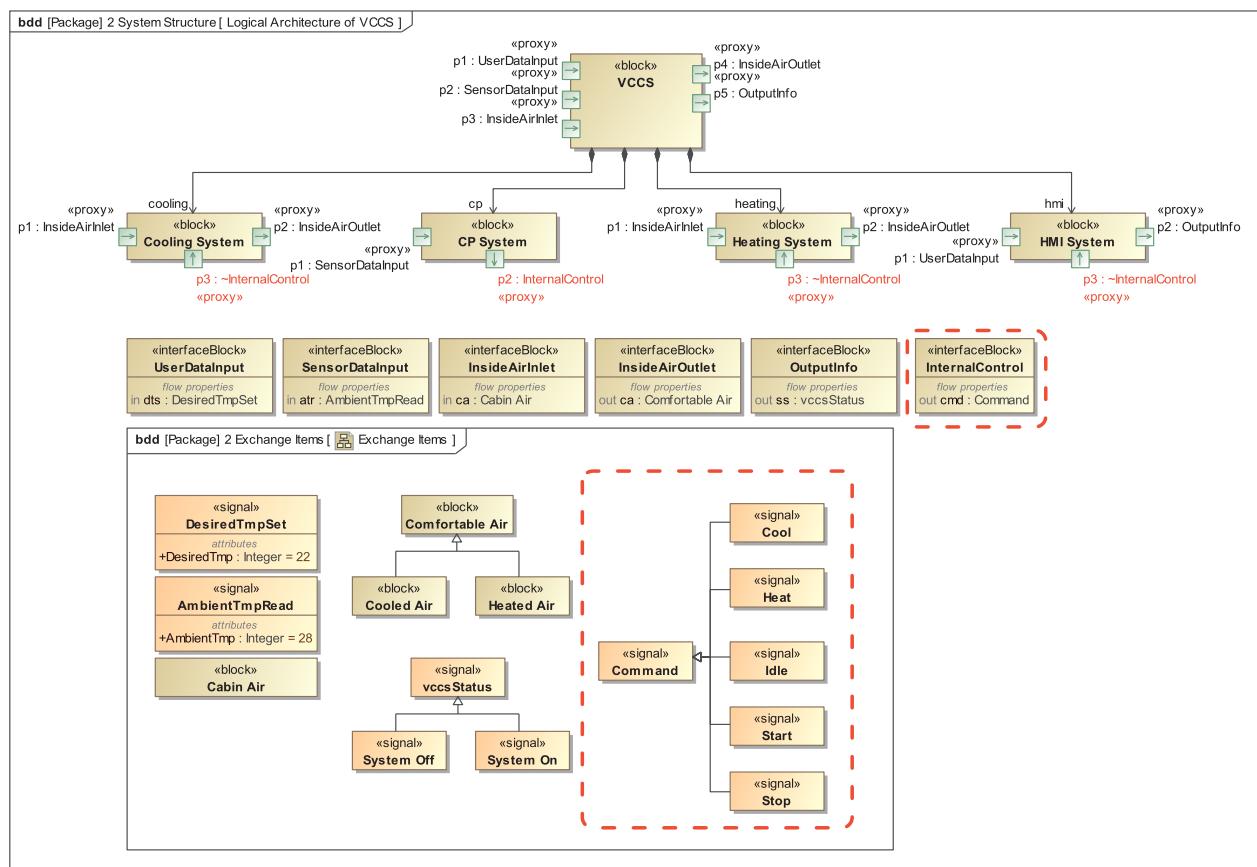


Note that the *cmd* flow property direction of the newly created interface block is *out*. This is because it was specified from the point of view of the CP System, which shall be able to send commands to other subsystems of the VCCS.

The next step is to specify that the *CP System* block is able to send the *Command* signal or any of its sub-types out, and the *Cooling System*, *Heating System*, and *HMI System* blocks are able to accept them. For this, you should create a proxy port for each block that captures the logical subsystem and then make the proxy ports of the *Cooling System*, *Heating System*, and *HMI System* blocks conjugated. As a result, names of the interface block typing these proxy ports appear decorated with a tilde ("~"), and its flow property directions become reversed for these proxy ports, which conveys that the *Cooling System*, *Heating System*, and *HMI System* blocks are able to accept the *Command* signal or any of its sub-types.

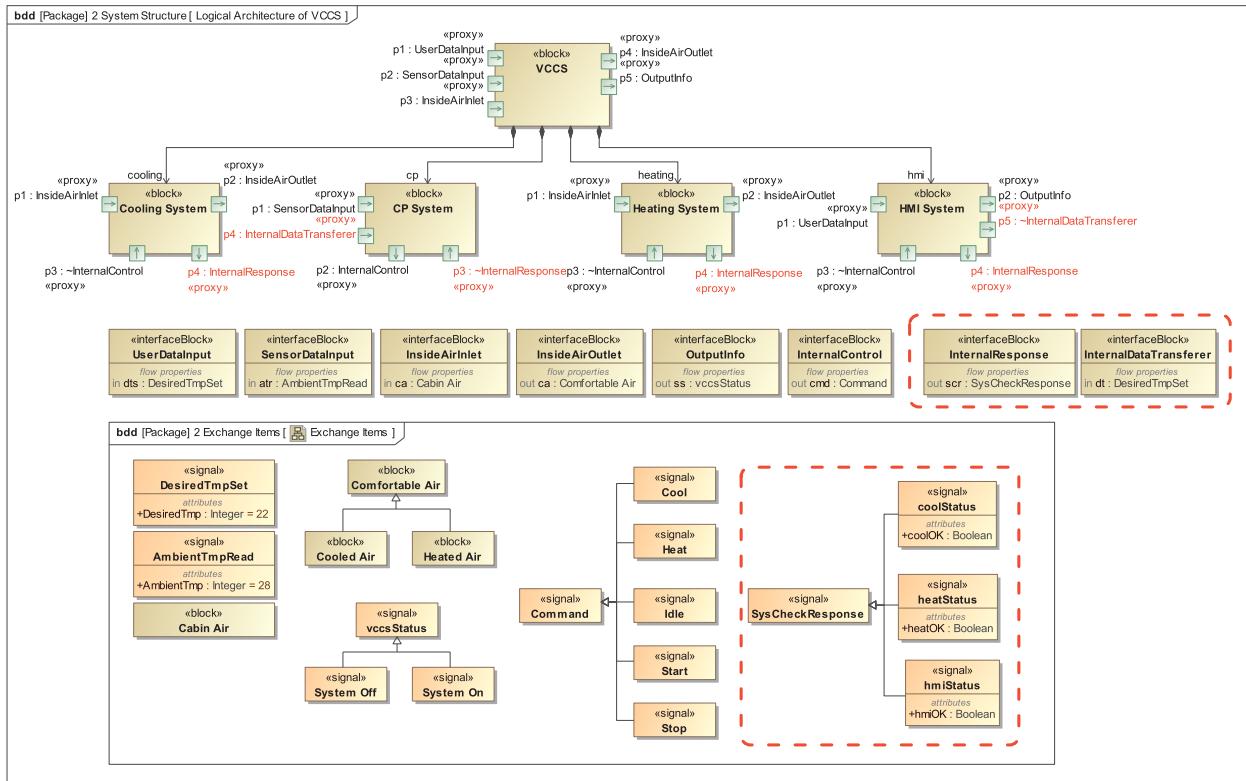
To make the proxy port conjugated

- Right-click that proxy port and then click to select **Is Conjugated**.



Info It's important to note here that even though this way of specifying compatible proxy ports is easier, it is not always best. It is more common in real-world projects to define a couple of compatible interface blocks instead of defining a single interface block and then making one of the proxy ports conjugated.

There are two more interfaces, the *InternalResponse* and *InternalDataTransferer*, that can be specified for internal communication. You can see them in the following figure. We assume that you capture all the highlighted SysML elements in your model.



System Structure done. What's next?

- Having logical subsystems and knowing how they will communicate, you can start working on their logical architecture models. Therefore, you could move to Chapter [Building the logical architecture of subsystems](#).
- However, you still have two cells left for creating the logical system architecture model. Therefore, you should proceed to the [System Behavior](#) and [System Parameters](#) cells.

System Behavior

What is it?

The LSA is not only about the system structure; the systems architect can capture the behavior of the whole **Sol** in the LSA model, too. However, this is not a common practice. Since the precise logical architecture of the **Sol** is not yet defined, the system behavior model appears too abstract and cannot say much about the system behavior. Moreover, it may conflict with the system behavior in the integrated model of the selected system configuration (see Chapter [Building system configuration model](#)). That's why this cell is often skipped when building the LSA of the **Sol** in real-world projects.

Unlike that in the LSA model, the system behavior in the integrated model of the selected system configuration is crucially important. Since the integrated model includes the architectures of all logical subsystems of the **Sol**, it can be used to analyze the integrated behavior of the entire system and perform the system requirements verification.

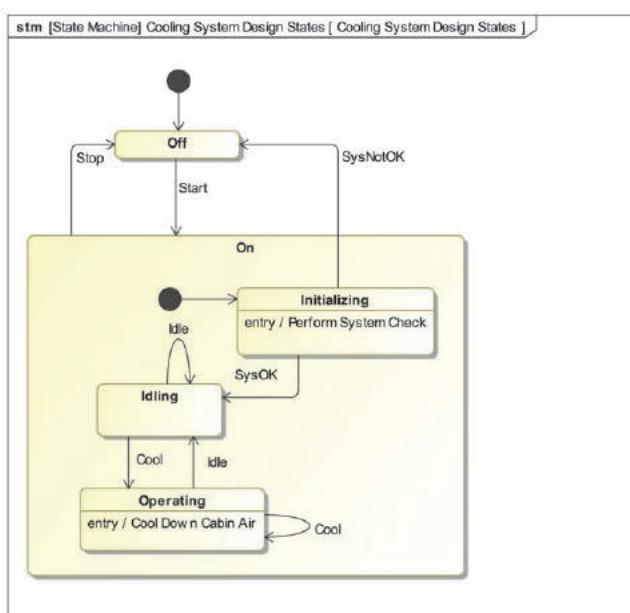
Who is responsible?

System behavior in the LSA model can be captured by the Systems Architect or Systems Engineer. He/she must be the head of the whole systems engineering project, the one who has "a big picture view" of the **Sol** and is responsible for the smooth integration of the subsystems and their components into the whole.

How to model?

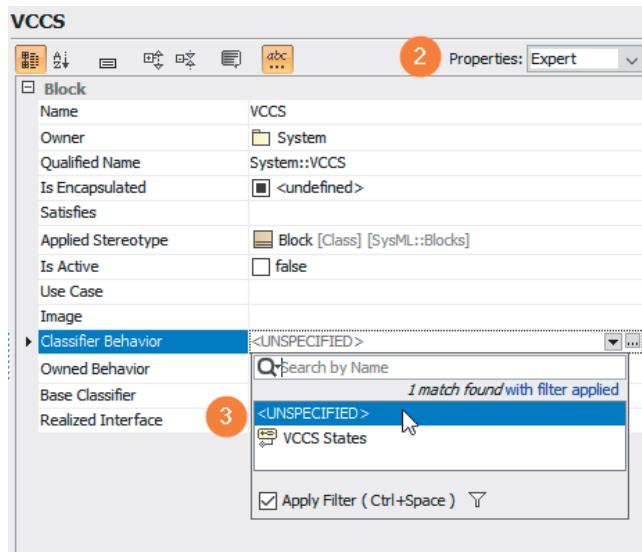
The system behavior in the LSA model can be captured by utilizing the infrastructure of the SysML state machine and activity or sequence diagrams in combination. The SysML state machine allows capturing states of the **Sol** and transitions between them in response to event occurrences over time. The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

Though the diagram in the following figure captures the behavior of the Cooling System, which is one of subsystems, the behavior of the **Sol** in the LSA model looks very much like this.



i To prevent conflicts with the system behavior in the integrated model of the selected system configuration, the system behavior defined in the LSA model must be unassigned as the classifier behavior from the block, which captures the **Sol**. For this, do the following:

1. Right-click the **VCCS** block and select **Specification**.
2. In the open dialog, from the **Properties** drop-down list, select **Expert** to extend the list of properties.
3. Click the cell of the **Classifier Behavior** property value and select **<UNSPECIFIED>** from the drop-down list.



Tutorial

To follow the common practice, we skip this cell in the LSA model and get back to it in the system configuration model (see Chapter [System Configuration Behavior](#)).

What's next?

The next cell to visit is [System Parameters](#).

System Parameters

What is it?

You can go to this cell as soon as the system structure is captured in the model, even as abstract as in the LSA model.

System parameters capture quantitative characteristics of the **system of interest**, that you can measure. They are used to calculate MoEs identified during the problem domain analysis (see Chapter [Measures of Effectiveness](#)). MoEs captured in the solution domain model satisfy non-functional quantitative system requirements.

Mathematical expressions for MoEs derivation from system parameters should also be specified in this cell. Knowing how to calculate MoEs enables early system requirements verification. Using the capabilities of the **modeling tool**, the model can be executed to calculate these values and show whether quantitative requirements are satisfied or not.

It's important to note here that not all MoEs in the solution domain must be based on the MoEs in the problem domain model. Some of them can be identified only when building the logical architecture of the [Sol](#).

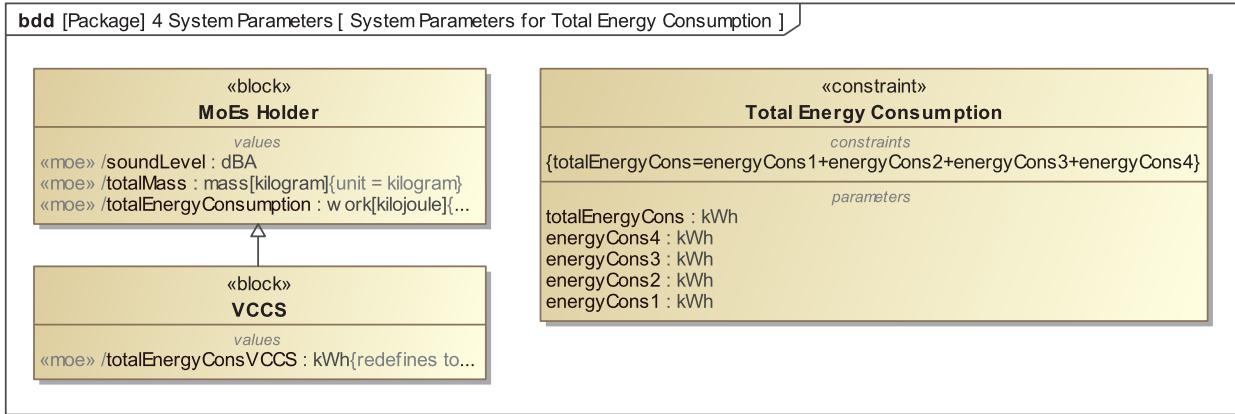
Who is responsible?

System parameters in the LSA model can be captured by the Systems Architect or Systems Engineer. He/she must be the head of the whole systems engineering project, the one, who has "a big picture view" of the [Sol](#) and is responsible for the smooth integration of the subsystems and their components into the whole.

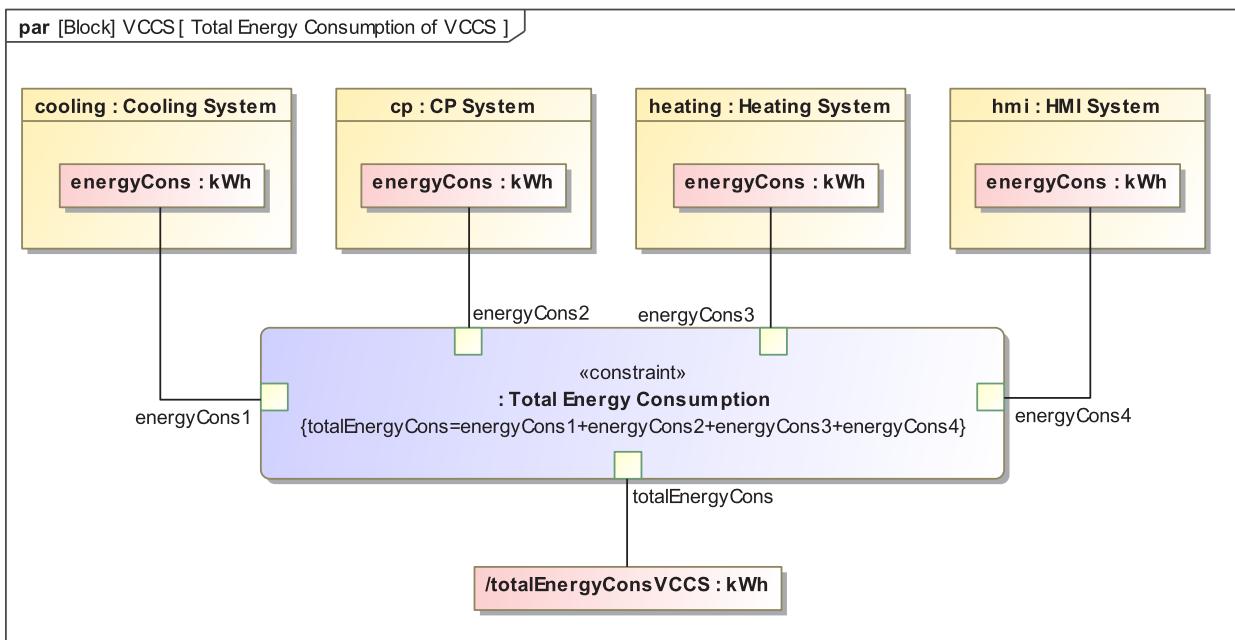
How to model?

As you should already know, measures of effectiveness can be captured in the model as value properties with the «moe» stereotype applied. These value properties must belong to the block, which represents the **system of interest**. You must also know that measures of effectiveness can be inherited from the reusable sets of MoEs, called MoEs holders, instead of specifying them from scratch.

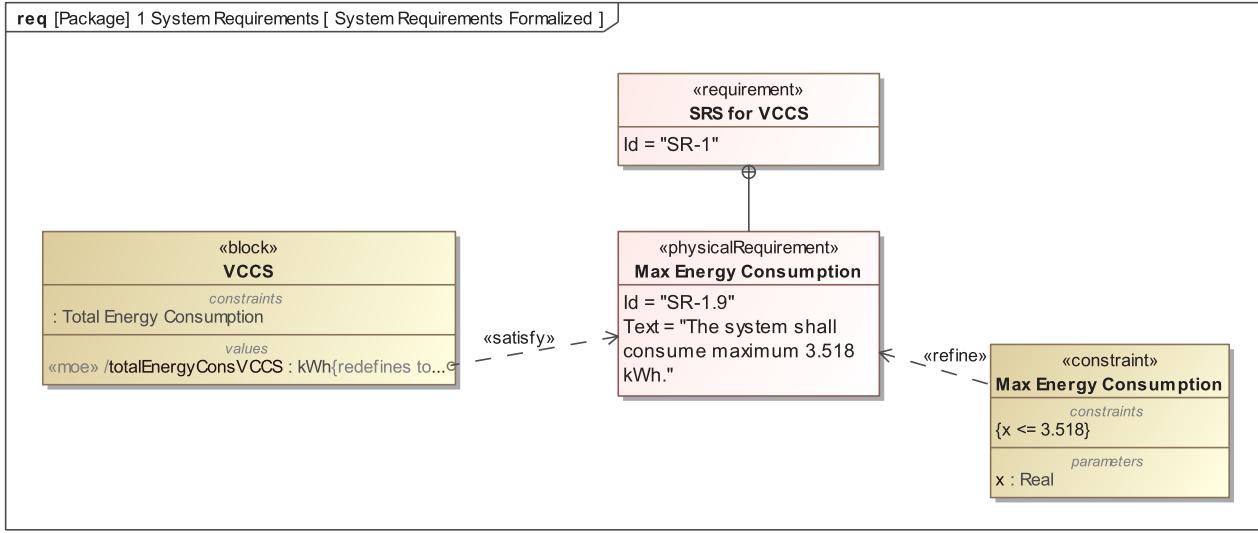
The formula for calculating the MoE can be captured as a constraint expression of some constraint block. The infrastructure of the SysML block definition diagram can be utilized for capturing MoEs, constraint expressions, and even required system parameters. The following figure displays the diagram that captures the *totalEnergyConsVCCS* measure of effectiveness and the formula for calculating this MoE as the constraint expression of the *Total Energy Consumption* constraint block.



To empower the constraint expression to perform calculations, you have to specify what system parameters should be consumed as variables of that constraint. Just like MoEs, system parameters can be captured in the model as value properties of the block that captures the **Sol** or any of its subsystems, only without any stereotype. The SysML parametrics diagram can be used to relate the system parameters and MoEs to the constraint variables, with system parameters as inputs and MoEs as outputs. Binding connectors (represented as solid lines in the following figure) should be used for this.



The simulation capabilities of the **modeling tool** allows you to calculate MoEs with given inputs. With the MoE calculated, you can verify the appropriate non-functional system requirement and show 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 create a satisfy relationship from the value property, which captures the MoE, to the appropriate system requirement. The following figure displays an example of the satisfy relationship between the *totalEnergyConsVCCS* value property and the *Max Energy Consumption* non-functional requirement, which means that the requirement is satisfied when the value of the *totalEnergyConsVCCS* value property is not greater than 3.518 kWh. As you can see, the natural language expression in the requirement text is refined by another constraint expression of the *Max Energy Consumption* constraint block with the $x \leq 3.518$ inequality.



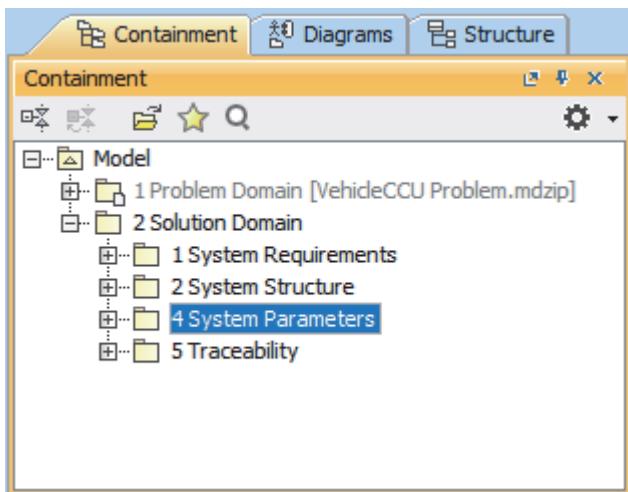
Although both satisfy and refine relationships are represented in the SysML requirements diagram, the bdd can also be used for this purpose.

Tutorial

- Step 1. Organizing the model for system parameters
- Step 2. Specifying the MoE for capturing the total energy consumption
- Step 3. Defining a method for calculating the total energy consumption
- Step 4. Specifying system parameters for calculating the total energy consumption
- Step 5. Binding constraint parameters to corresponding value properties
- Step 6. Performing early system requirements verification
- Step 7. Storing values in the model

Step 1. Organizing the model for system parameters

As the system requirements specification defines, we need to specify the mathematical expression for calculating the total energy consumed by the Vehicle Climate Control System. The constraint block which captures this expression can be stored in a separate package within the 2 Solution Domain package.



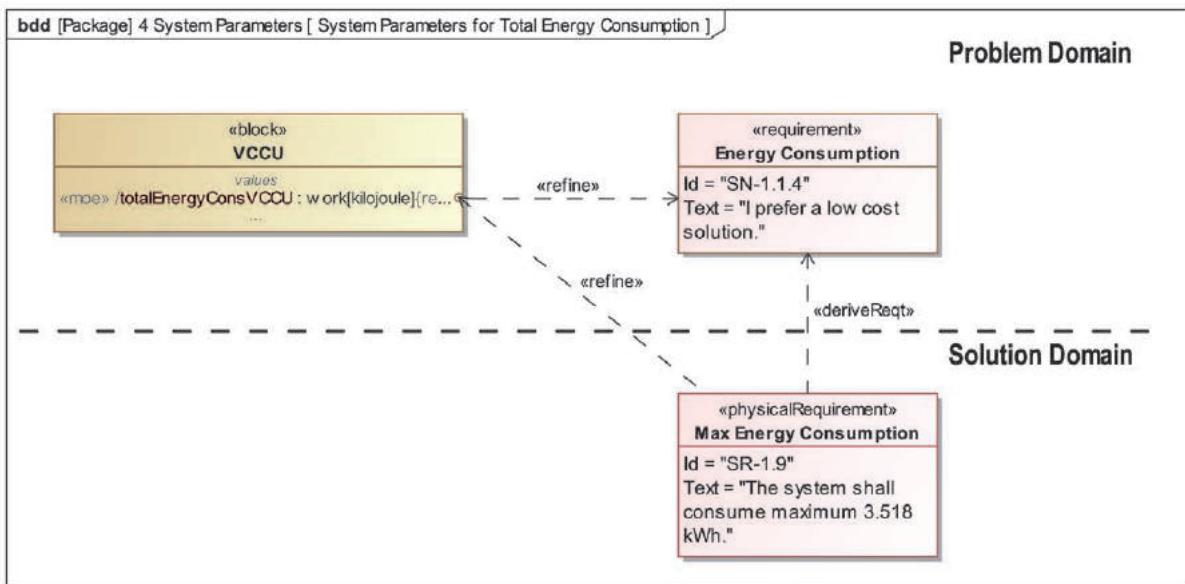
To organize the model for storing constraint blocks

1. Open the solution domain model (the *VehicleCCS Solution.mdzip* file) you created in Chapter [System Requirements](#), 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 *4 System Parameters* to specify the name of the new package and press Enter.

i Even though we don't have the *3 System Behavior* package (remember that it's not necessary to model the system behavior in the LSA model), we skip "3" in package numbering to correspond to the layout of the MagicGrid framework.

Step 2. Specifying the MoE for capturing the total energy consumption

As you can see in the following figure, the system requirements specification (see Chapter [System Requirements](#)) and the appropriate MoE (see Chapter [Measures of Effectiveness](#)) in the problem domain model determine the specification of the MoE that captures the total energy consumed by the VCCS, in the solution domain model.



It was already mentioned that MoEs can be specified from scratch or inherited from the MoEs holder. The following procedure describes inheritance from the MoEs holder. The inherited MoE is redefined to change its name and type. Redefinition is necessary for automated requirements verification, which is performed in one of the following steps.

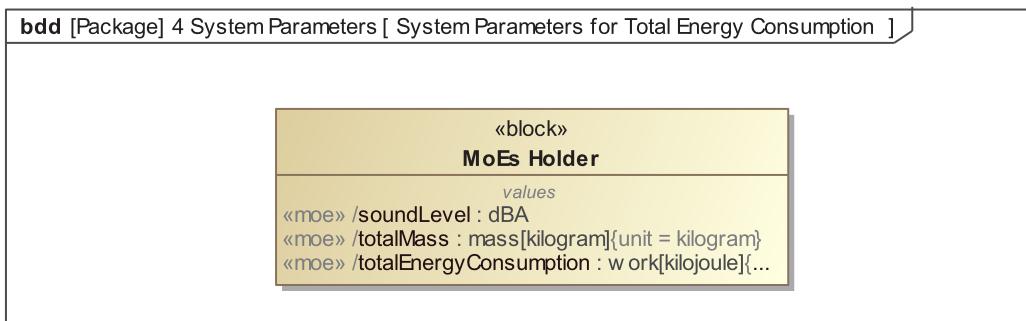
To specify the MoE for capturing the total energy consumed by the VCCS

1. Create a bdd for capturing system parameters:

- a. Right-click the *4 System Parameters* package you created in [step 1 of this cell tutorial](#) and select **Create Diagram**.
- b. In the search box, type *bdd* (the acronym of the SysML block definition diagram), and then double-press Enter. The diagram is created.
- c. Type *System Parameters for Total Energy Consumption* to specify the name of the new diagram and press Enter again.

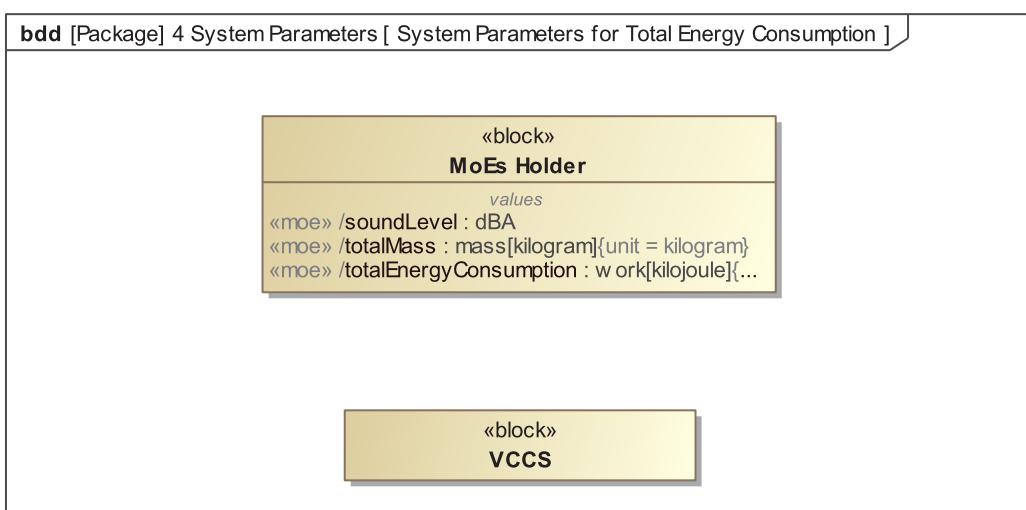
2. Display the *MoEs Holder* block on the diagram:

- a. Press Ctrl + Alt + F to open the **Quick Find** dialog.
- b. In the dialog, type *moe*s.
- 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.



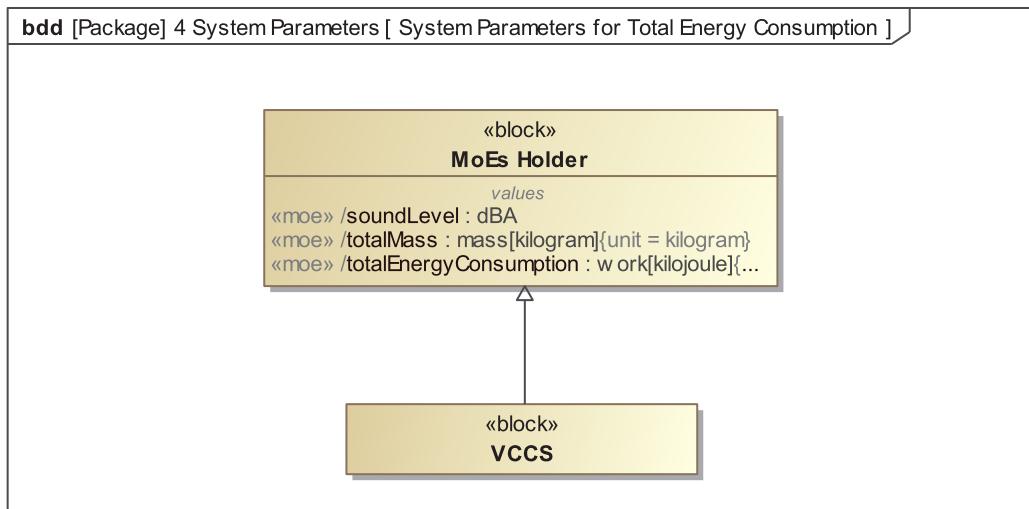
3. Display the *VCCS* block on the diagram:

- a. Press Ctrl + Alt + F to open the **Quick Find** dialog.
- b. In the dialog, type *vccs*.
- c. When you see the *VCCS* block selected in the search results list, press Enter. The block is selected in the Model Browser.
- d. Drag the *VCCS* block to the diagram pane. The shape of the block is displayed on the diagram, too.



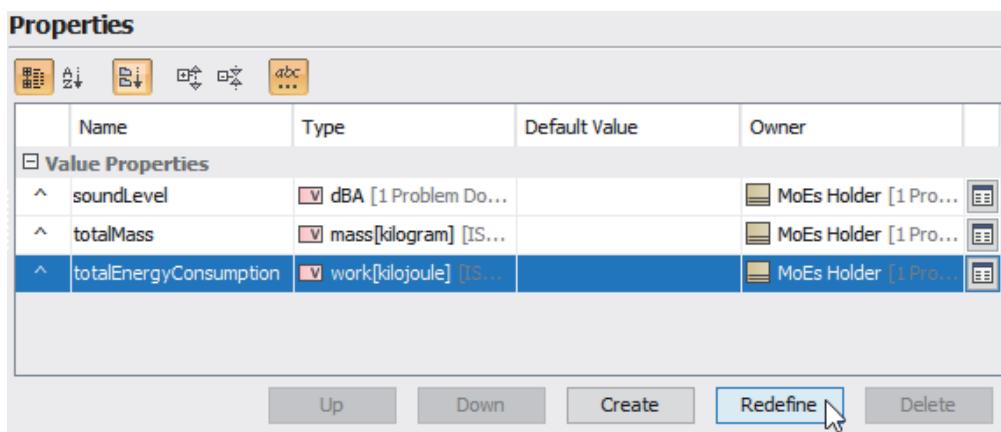
4. Establish the generalization between the *VCCS* block and the *MoEs Holder* block:

- Select the shape of the *VCCS* block and click the Generalization button  on its smart manipulator toolbar.
- Select the shape of the *MoEs Holder* block. The *MoEs Holder* block becomes the super-type of the *VCCS* block, which means that the *VCCS* block inherits all MoEs from the *MoEs Holder* block.



5. Redefine the *totalEnergyConsumption* value property for the *VCCS* block:

- Double click the *VCCS* block to open its Specification.
- On the left of the open dialog, click **Properties**.
- Select the *totalEnergyConsumption* value property and click the **Redefine** button below.



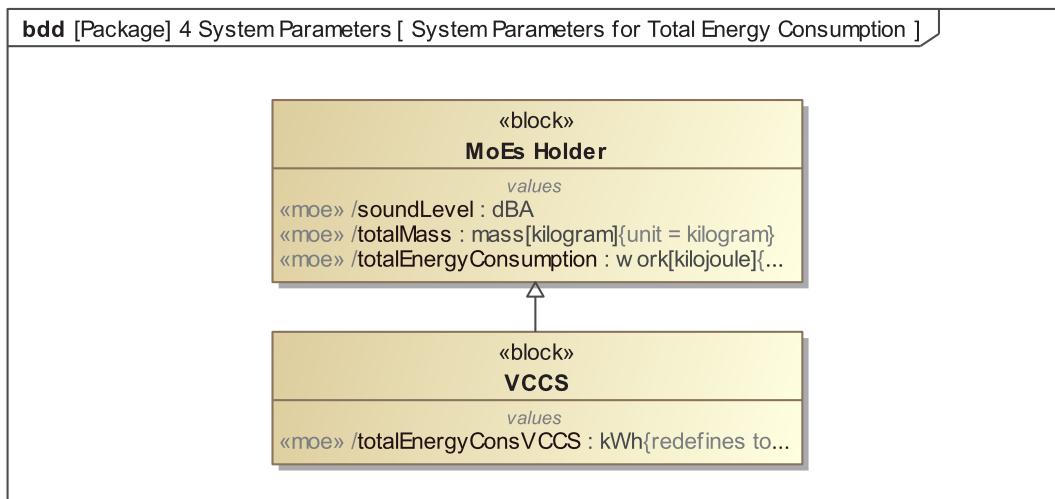
- In the **Type** cell, immediately type *kWh* and press Enter. The *kWh* value type is created in the model and set as type for the redefined value property.

- e. In the **Name** cell, type *totalEnergyConsVCCS* to rename the redefined value property press Enter.

Properties					
	Name	Type	Default Value	Owner	
Value Properties					
	totalEnergyConsVCCS	kWh [2 Solution Do...]		VCCS [2 Solution ...]	
^	soundLevel	dBA [1 Problem Do...]		MoEs Holder [1 Pro...]	
^	totalMass	mass[kilogram] [IS...]		MoEs Holder [1 Pro...]	

- f. Click **Close**.

When you're done, the *System Parameters for Total Energy Consumption* bdd in your model should be similar to the one below.



Step 3. Defining a method for calculating the total energy consumption

The next thing you need to do is capture the mathematical expression which specifies how to calculate a few minutes ago specified MoE. It can be captured as the constraint expression of some constraint block stored in your model.

Let's say the total energy consumption of the VCCS equals a sum of energies consumed by all its subsystems. The constraint expression can be defined as follows:

$$\text{totalEnergyCons} = \text{energyCons1} + \text{energyCons2} + \text{energyCons3} + \text{energyCons4}$$

where:

1. *totalEnergyCons* stands for the total energy consumed by the VCCS
2. *energyCons1* may stand for the energy consumed by the Cooling System
3. *energyCons2* may stand for the energy consumed by the CP System
4. *energyCons3* may stand for the energy consumed by the Heating System
5. *energyCons4* may stand for the energy consumed by the HMI System

In terms of SysML, *totalEnergyCons*, *energyCons1*, ..., and *energyCons4* are generally referred to as constraint parameters and later should be bound to value properties.

A bdd can be utilized for capturing constraint blocks. This constraint block is not an exception; you can use the bdd created in [step 2 of this cell tutorial](#).

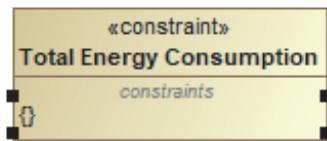
To capture the method for calculating the total energy consumed by the VCCS

1. Create a constraint block:

- Open the *System Parameters for Total Energy Consumption* diagram, if not opened yet.
- Click the **Constraint Block** button on the diagram palette and then click an empty space in that diagram pane. An unnamed constraint block is created.
- Type *Total Energy Consumption* 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:

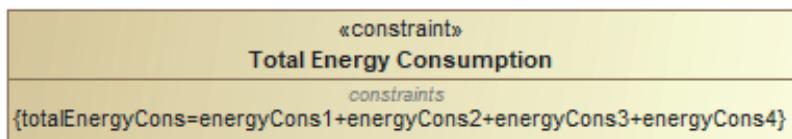
- Click the symbol of the empty constraint expression on the *Total Energy Consumption* constraint block.



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

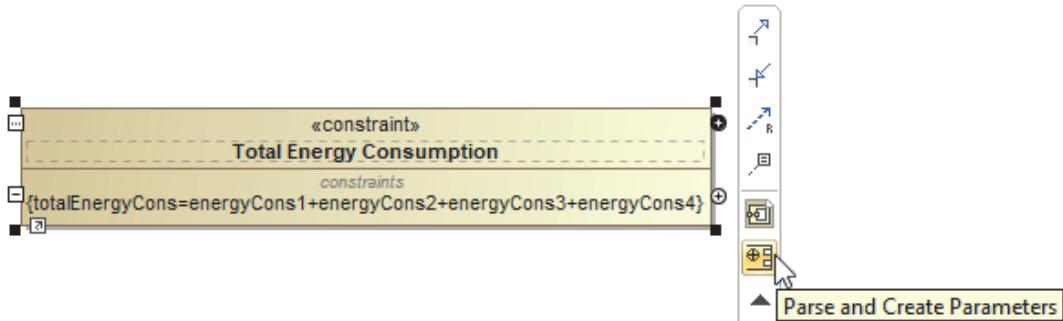


- Type *totalEnergyCons=energyCons1+energyCons2+energyCons3+energyCons4* and then press Enter.



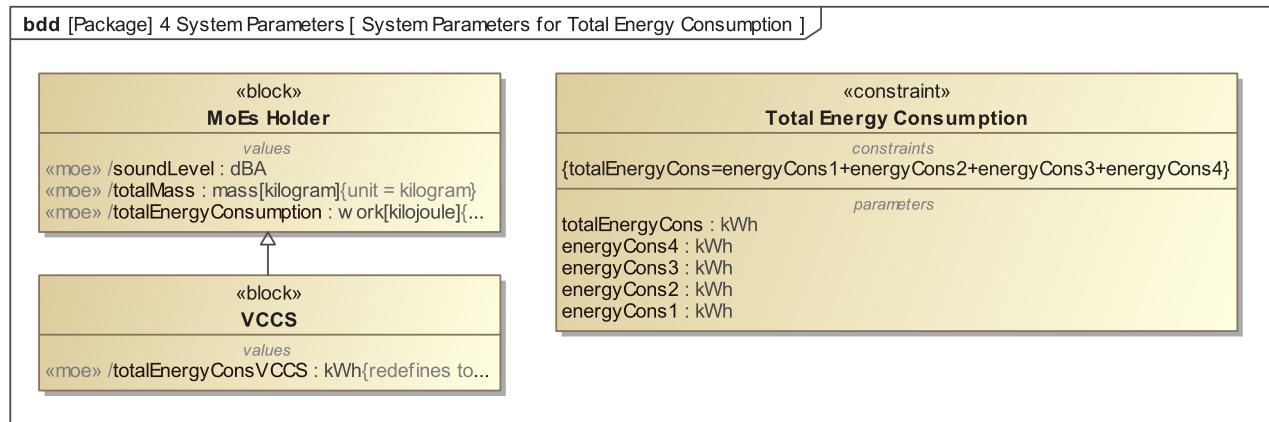
3. Specify constraint parameters:

- Select the shape of the *Total Energy Consumption* 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 *kWh* directly on the shape of the constraint block:
- Double-click the type to select it, type *kWh*, and press Ctrl + Spacebar.
 - Select the *kWh* type from the list below.
 - Press Enter.

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



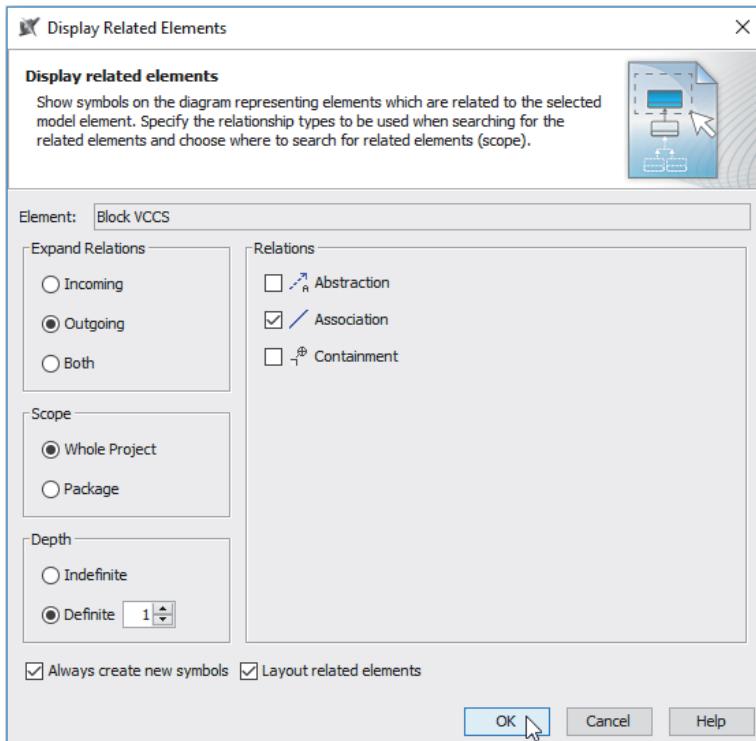
Step 4. Specifying system parameters for calculating the total energy consumption

With the constraint expression, you can easily determine what system or (in this case) subsystem parameters are necessary for calculating the total energy consumed by the VCCS. Input parameters of the *Total Energy Consumption* constraint block can help you with this task.

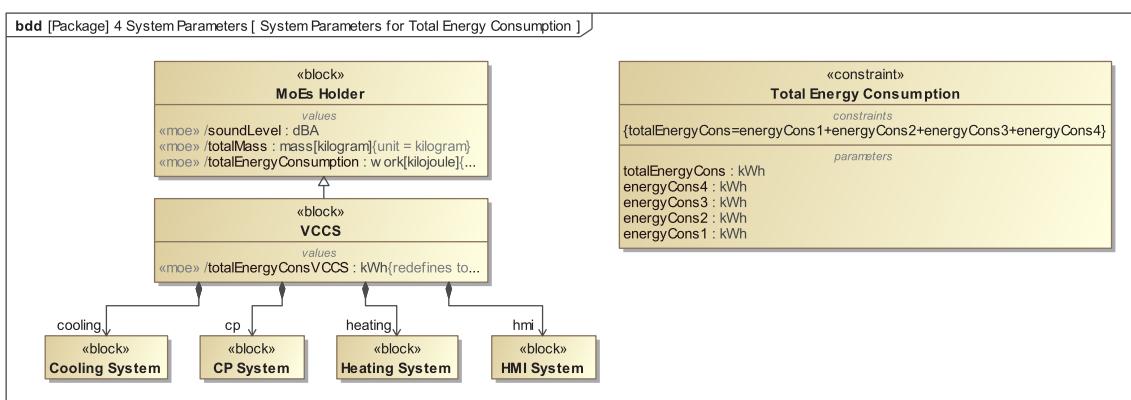
Remember that it has four input parameters, one each for the energy consumption of a single subsystem of the VCCS, and requires four subsystem parameters in your model. Appropriate value properties should be specified for each block that captures one of the subsystems of the VCCS.

To specify subsystem parameters for capturing the energy consumed by each subsystem of the VCCS

1. Open the *System Parameters for Total Energy Consumption* diagram, if not opened yet.
2. Display the blocks capturing the logical subsystems of the VCCS block:
 - a. Right-click the VCCS block and select **Display > Display Related Elements**.
 - b. In the open dialog, click **Outgoing** and then click to clear the **Abstraction** check box.



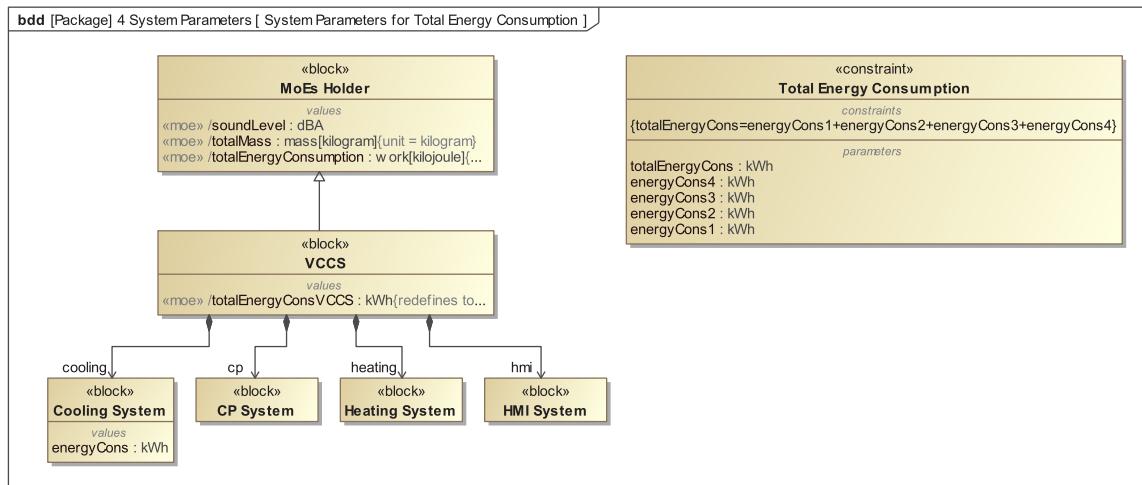
- c. Click **OK** to close the dialog. As a result, all the blocks representing the subsystems of the VCCS are displayed on the diagram pane.



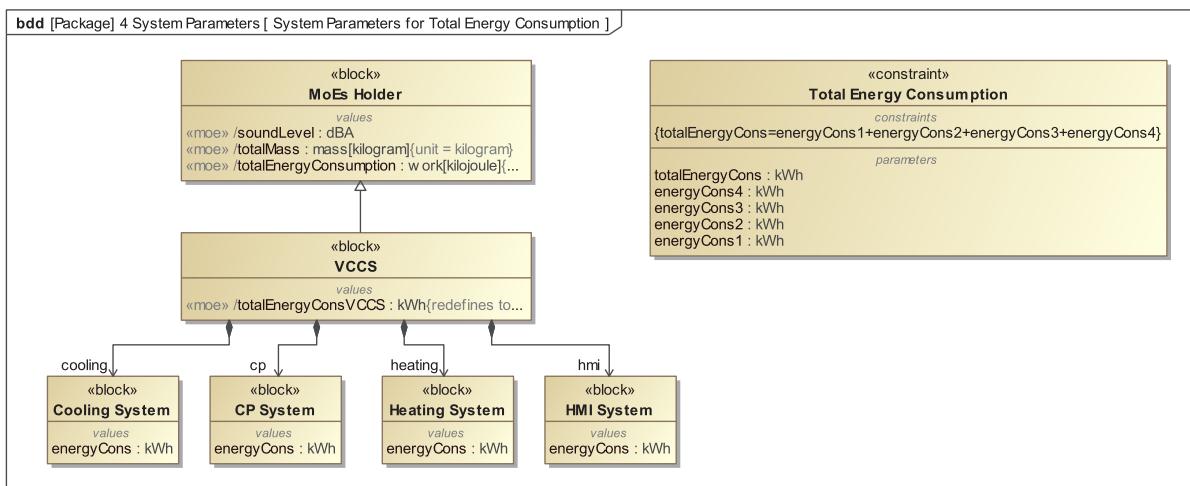
3. Create the *energyCons* value property for the *Cooling System* block:

- a. Select the shape of the *Cooling System* block, click the Create Element button **+**, and then select **Value Property**.
- b. Type *energyCons:kW* and press Ctrl + Spacebar to select the suitable value type.
- c. Select the *kWh* value type from the list below and press Enter.

d. Press Enter again.



4. Repeat step 3 to create value properties for the *CP System*, *Heating System*, and *HMI System* blocks.



Step 5. Binding constraint parameters to corresponding value properties

Once you have all the necessary value properties in the model, it's time to bind them to the appropriate constraint parameters. 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* block. However, creating the SysML parametrics diagram is not enough to get started. Establishing binding connectors is possible only after the constraint property typed by the *Total Energy Consumption* constraint block is created for the *VCCS* block. Be aware that the same constraint block can be used in multiple contexts, and binding connectors are context-specific relationships.

To bind the constraint parameters to the corresponding value properties

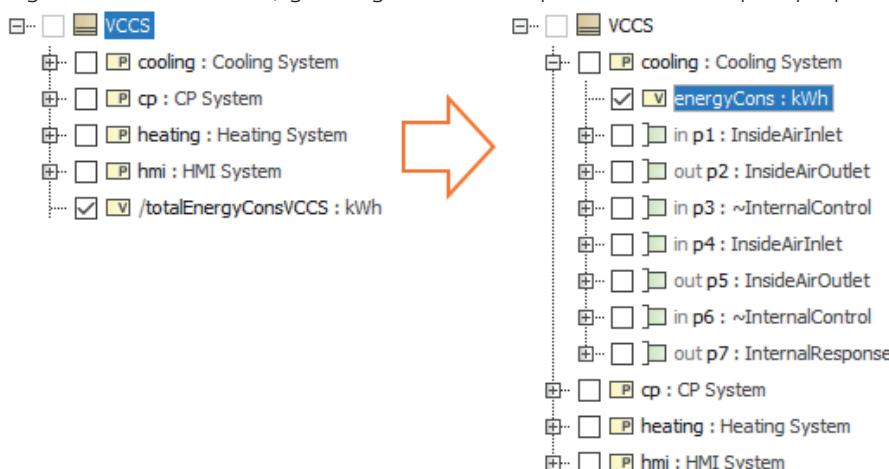
1. Select the **VCCS** block in the Model Browser:

- Press Ctrl + Alt + F. The **Quick Find** dialog opens.
- Type **vccs**.
- When you see the **VCCS** block selected in the search results list, press Enter.

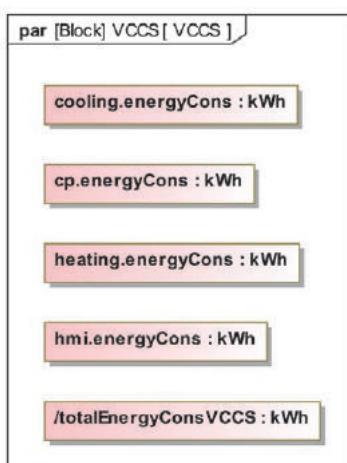
2. Create a SysML parametrics diagram for the **VCCS** block:

- Press Ctrl + N. The **Create Diagram** dialog opens.
- 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/Properties** dialog opens.
- Besides the **totalEnergyConsVCCS** value property (which is selected by default), select to display four more value properties, each named **energyCons** and owned by the block which captures a single subsystem of the VCCS.

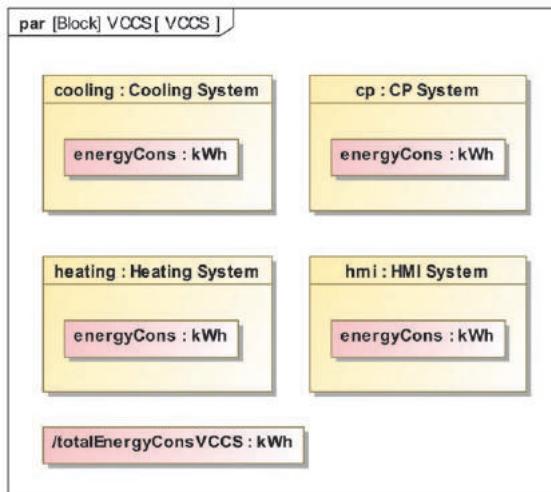
 If you cannot see them, you might need to expand the related part properties.



- 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.



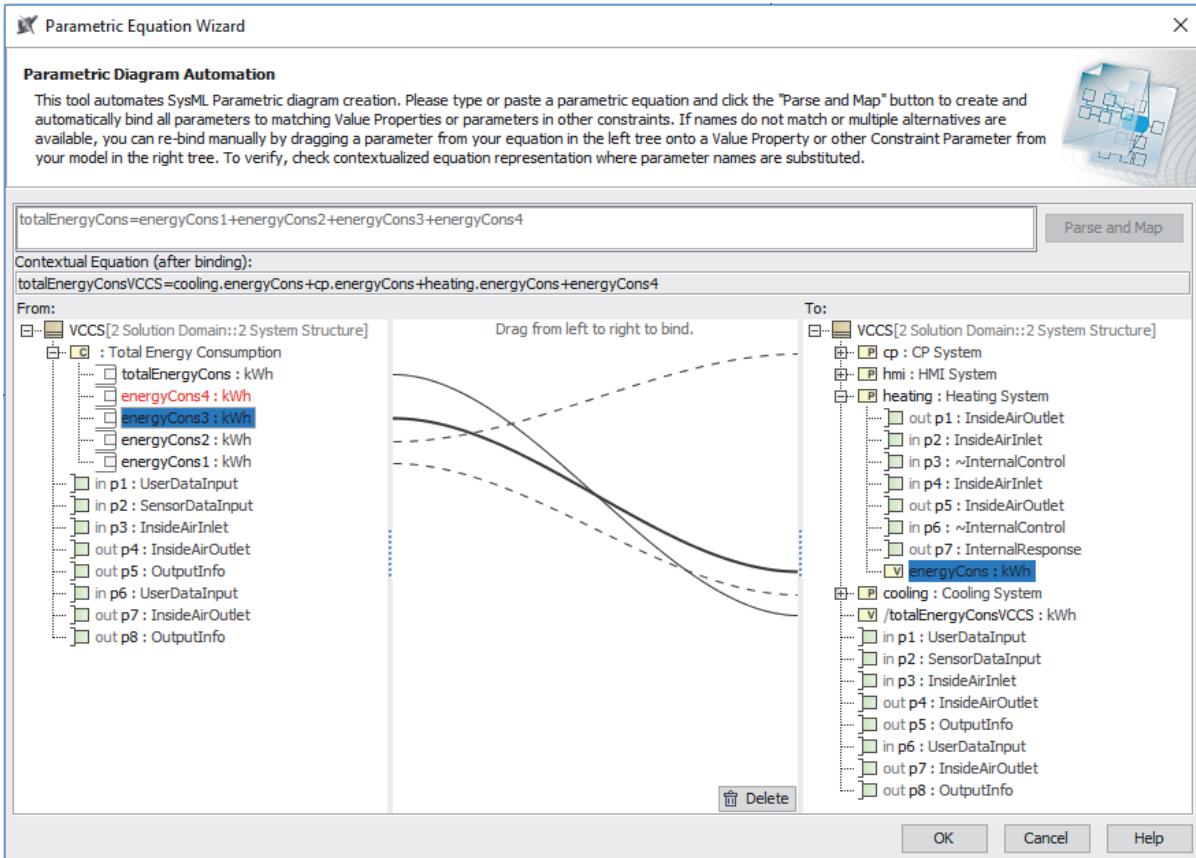
- i** As you can see in the figure above, the deeply nested value properties are displayed in the dot notation. If you want to switch to the nested notation, select all the shapes, except the shape for the `totalEnergyConsVCCS` value property, then right-click them and select Refactor > Convert to Nested Parts.



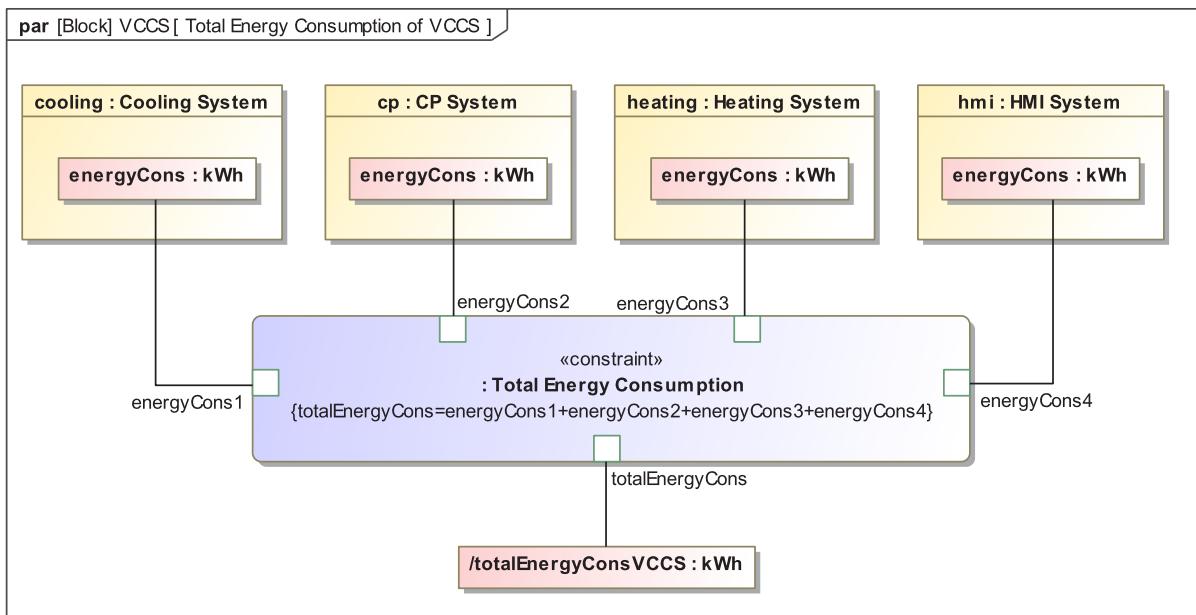
e. Type *Total Energy Consumption of VCCS* to name the diagram and press Enter.

3. Find the *Total Energy Consumption* constraint block:
 - a. Press Ctrl + Alt + F. The **Quick Find** dialog opens.
 - b. Type *total en*.
 - c. When you see the *Total Energy Consumption* constraint block selected in the search results list, press Enter. The constraint block is selected in the Model Browser.
4. Drag the constraint block to the *Total Energy Consumption of VCCS* diagram. An unnamed constraint property, typed by the *Total Energy Consumption* constraint block, is created for the *VCCS* block. The **Parametric Equation Wizard** dialog opens concurrently.
5. 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):
 - *energyCons1* to the *energyCons* value property of the *Cooling System* block
 - *energyCons2* to the *energyCons* value property of the *CP System* block
 - *energyCons3* to the *energyCons* value property of the *Heating System* block
 - *energyCons4* to the *energyCons* value property of the *HMI System* block

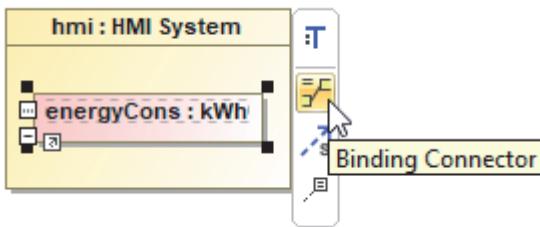
- i**
- You may need to expand the structure on the right side of the dialog to see the value property you need. Remember that they belong to the subsystem blocks, but not to the system block directly.
 - Once bound, the constraint parameter changes its color from red to grey.



6. After you bind all the items, click **OK**. The dialog closes and all the bindings are displayed on the diagram.



Binding connectors can also be created from the smart manipulator toolbar of the value property (see the following figure) or constraint parameter.



Now you can execute the model, provide input values, and see the result of the calculation. To start the execution, click the Run button in the toolbar above the *Total Energy Consumption* of VCCS diagram. When in the **Variables** tab of the **Simulation** panel (by default, it is on the right side of the **Simulation** panel), you can specify the input values and see the output value (result of the calculation).

Name	Value
VCCS	VCCS@4e3d3591
/totalEnergyConsVCCS : kWh	3,8500
cooling : Cooling System	Cooling System@5731ca9f
energyCons : kWh	1,1000
cp : CP System	CP System@3e6a51d7
energyCons : kWh	0,7500
heating : Heating System	Heating System@5b951cb6
energyCons : kWh	1,5000
hmi : HMI System	HMI System@78feae94
energyCons : kWh	0,5000

Step 6. Performing early system requirements verification

Having the concrete value of the value property that captures the MoE, you can verify the relevant system requirement and show whether or not it is satisfied. The **modeling tool** enables you to perform this verification automatically. To get ready for the automated requirements verification, you need to do the following:

1. Formalize the constraint captured as the natural language phrase in the text of the particular system requirement.
2. Indicate the value property whose value determines whether or not the system requirement is satisfied.

In this case, you should formalize the *Max Energy Consumption* system requirement, which indicates that "The system shall consume maximum 3.518 kWh." This requires creating another constraint block with a constraint expression (this time the $x \leq 3.518$ inequality), and making it refine the relevant system requirement.

To formalize the *Max Energy Consumption* system requirement

1. Create a separate diagram for the system requirements formalization:

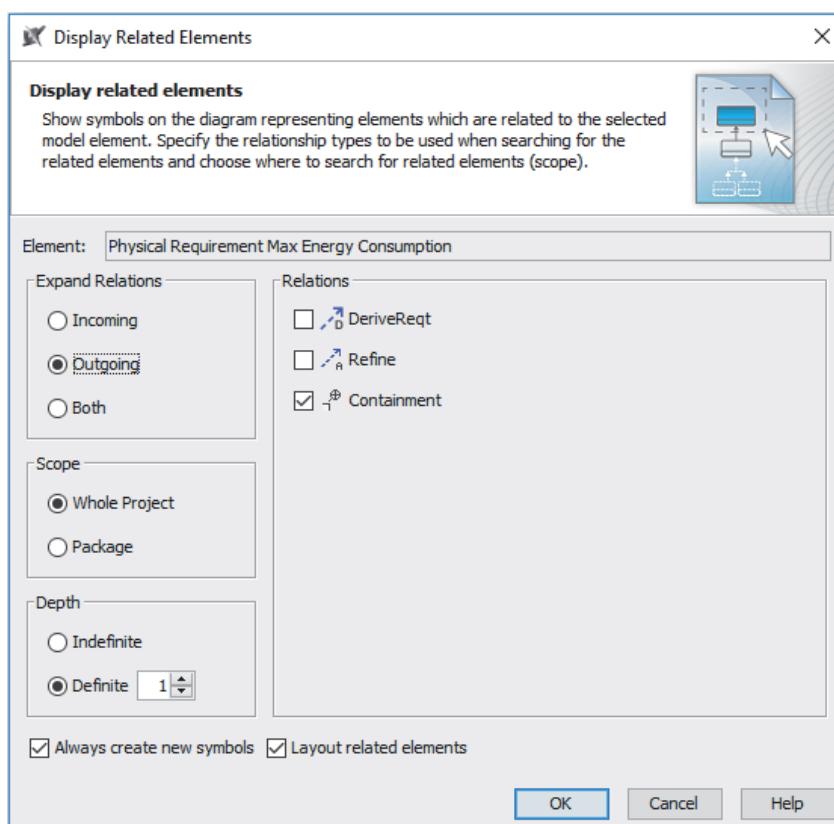
- a. Right-click the *1 System Requirements* package and select **Create Diagram**.
- b. In the search box, type *rd* (the acronym for the SysML requirements diagram), and then press Enter. The diagram is created.
- c. Type *System Requirements Formalized* to specify the name of the new diagram and press Enter again.

2. Display the *Max Energy Consumption* requirement on the diagram pane:

- a. Press Ctrl + Alt + F. The **Quick Find** dialog opens.
- b. Type *max en*.
- c. When you see the *Max Energy Consumption* requirement selected in the search results list, press Enter. The requirement is selected in the Model Browser.
- d. Drag the requirement to the newly created diagram pane.

3. Display the *SRS for VCCS* requirements on the diagram pane:

- a. Right-click the *Max Energy Consumption* requirement and select **Display > Display Related Elements**.
- b. In the open dialog, click **Outgoing**, click to clear the **DeriveReqt** and **Refine** check boxes, and then click to select the **Containment** check box.



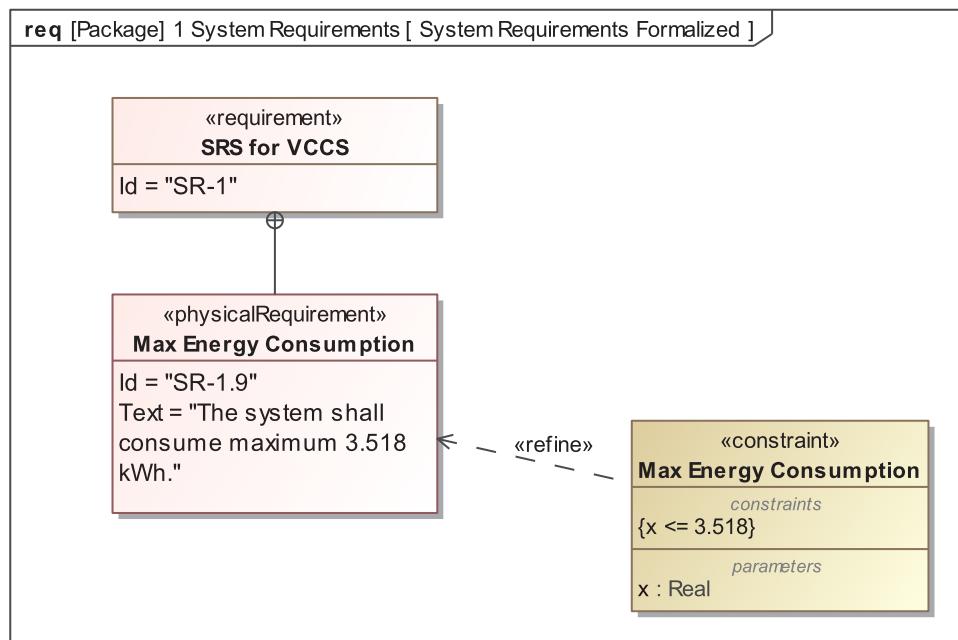
- c. Click **OK**.

4. Create the *Max Energy Consumption* constraint block:

- a. In the diagram palette, click the **Constraint Block** button.
- b. Click an empty place on the diagram pane.
- c. Immediately type *Max Energy Consumption* to name the newly created element and then press Enter.
- d. Specify the $x \leq 3.518$ inequality as the constraint expression:
 - i. Click the symbol of the empty constraint expression on the constraint block.
 - ii. Click the selection again. The empty constraint expression switches to the edit mode.
 - iii. Type $x \leq 3.518$ and then press Enter.
- e. Select the shape of the *Max Energy Consumption* constraint block and click the Parse and Create Parameters button  on its smart manipulator toolbar. The x constraint parameter is automatically extracted from the constraint expression and displayed in the *parameters* compartment box on the shape.
- f. Select the parameters and change their types from the default *Real* to *kWh* directly on the shape of the constraint block:
 - i. Double-click the type to select it, type *kWh*, and press Ctrl + Spacebar.
 - ii. Select the *kWh* type from the list below it.
 - iii. Press Enter.

5. Specify the refine relationship between the *Max Energy Consumption* constraint block and the relevant system requirement:

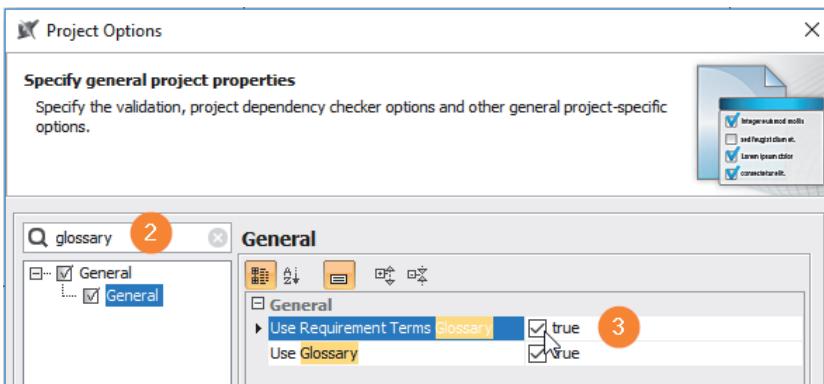
- a. Select the shape of the constraint block and click the Refine button  on its smart manipulator toolbar.
- b. Click the shape of the *Max Energy Consumption* system requirement. The refine relationship conveying that the textual system requirement is formalized by the *Max Energy Consumption* constraint block is created.



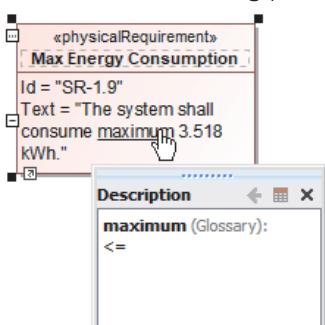
- i** The **modeling tool** enables you to create the constraint block automatically by parsing the requirement's text. First, you must update the project options.

To update project options

1. From the main menu, select **Options > Project**.
2. In search box of the **Project Options** dialog, type *glossary*.
3. Click to set the **Use Requirement Terms Glossary** option to *true*.
4. Click **OK** to close the dialog.



As a result, the word *maximum* is underlined on the shape of the *Max Energy Consumption* system requirement. If you hover the mouse over that text, you can see the inequality operator, which is automatically parsed from the word *maximum*.



The $x \leq 3.518$ inequality is stored in the model as the constraint expression of the virtual constraint block, which can be extracted on demand, as this is not necessary for automated requirements verification; the virtual constraint expression is enough.

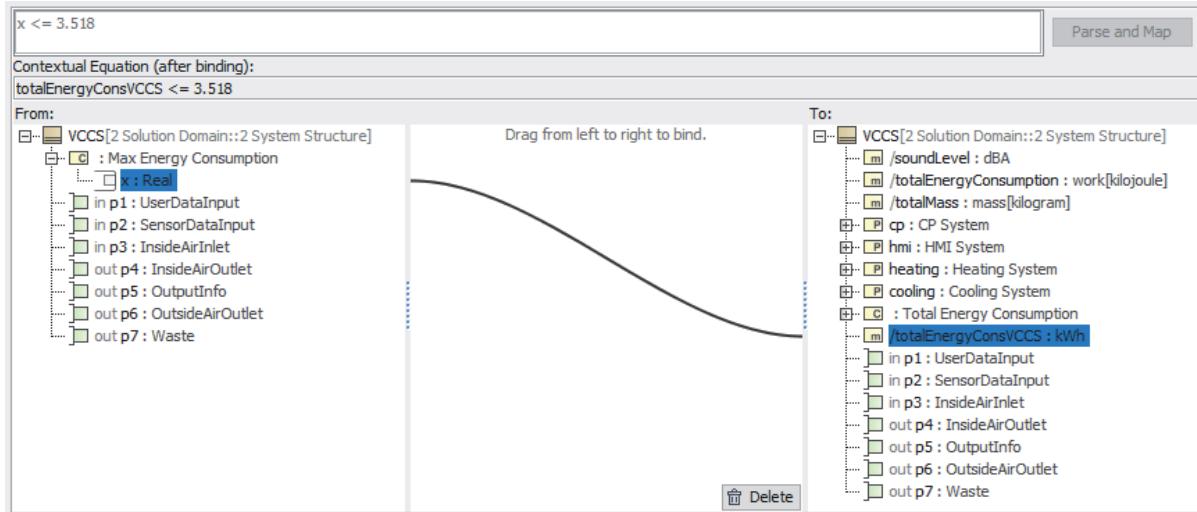
To extract the constraint block from the *Max Energy Consumption* system requirement

1. Right-click the requirement and select **Tools > Extract Constraint Block From Requirement**. The new constraint block is created in the model and automatically related to the *Max Energy Consumption* system requirement.
2. Drag the *Max Energy Consumption* constraint block onto the diagram pane. The shape of the constraint block and the refine relationship pointing from that constraint block to the *Max Energy Consumption* system requirement are displayed on the diagram.

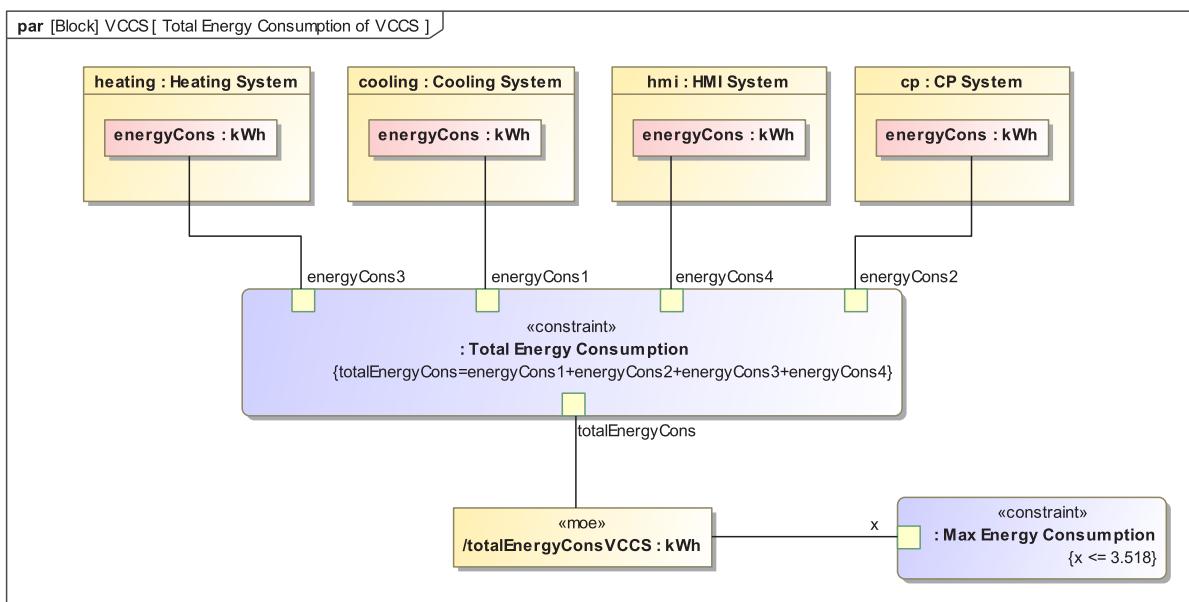
Next, specify that values of the *totalEnergyConsVCCS* value property can be used to determine whether or not the *Max Energy Consumption* system requirement is satisfied. For this, you need to bind the *x* parameter of the *Max Energy Consumption* constraint block to the *totalEnergyConsVCCS* value property. This can be done in the *Total Energy Consumption of VCCS* parametrics diagram.

To bind the *x* parameter of the *Max Energy Consumption* constraint block to the *totalEnergyConsVCCS* value property

1. Open the *Total Energy Consumption of VCCS* parametrics diagram.
2. Drag the *Max Energy Consumption* constraint block to the diagram pane.
3. In the open dialog, select the *x* constraint parameter on the left and drag it to the *totalEnergyConsVCCS* value property on the right.



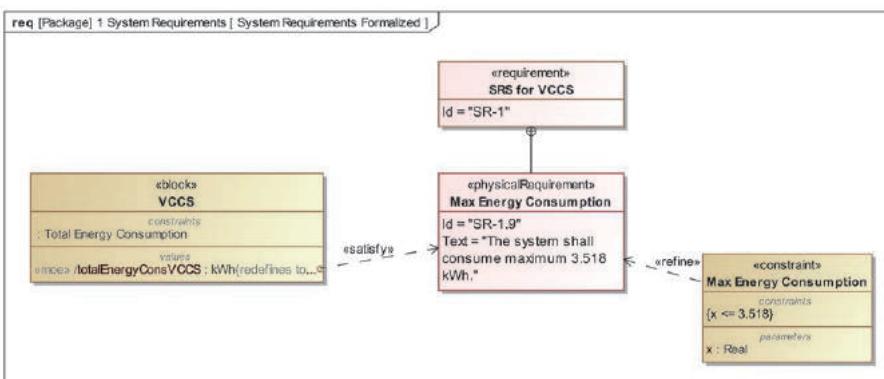
4. Click **OK**. The *Total Energy Consumption of VCCS* parametrics diagram is updated with the new constraint property for verification of the *Max Energy Consumption* system requirement.



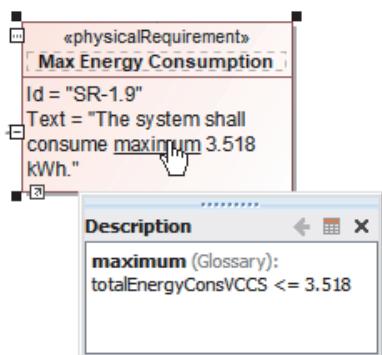
- i** If the **Use Requirement Terms Glossary** option is set to *true* in your project, you don't need to update the *Total Energy Consumption of VCCS* parametrics diagram. It will be enough to establish a satisfy relationship pointing from the *totalEnergyConsVCCS* value property to the *Max Energy Consumption* system requirement.

To create a satisfy relationship from the *totalEnergyConsVCCS* value property to the *Max Energy Consumption* system requirement

1. Display the *VCCS* block on the *System Requirements Formalized* diagram:
 - a. Press Ctrl + Alt + F. The **Quick Find** dialog opens.
 - b. Type *vccs*.
 - c. When you see the *VCCS* block selected in the search results list, press Enter. The block is selected in the Model Browser.
 - d. Drag the block to the diagram pane.
2. Select the *totalEnergyConsVCCS* value property (not the entire block!).
3. Click the Satisfy button  on the smart manipulator toolbar of that value property.
4. Click the *Max Energy Consumption* system requirement.

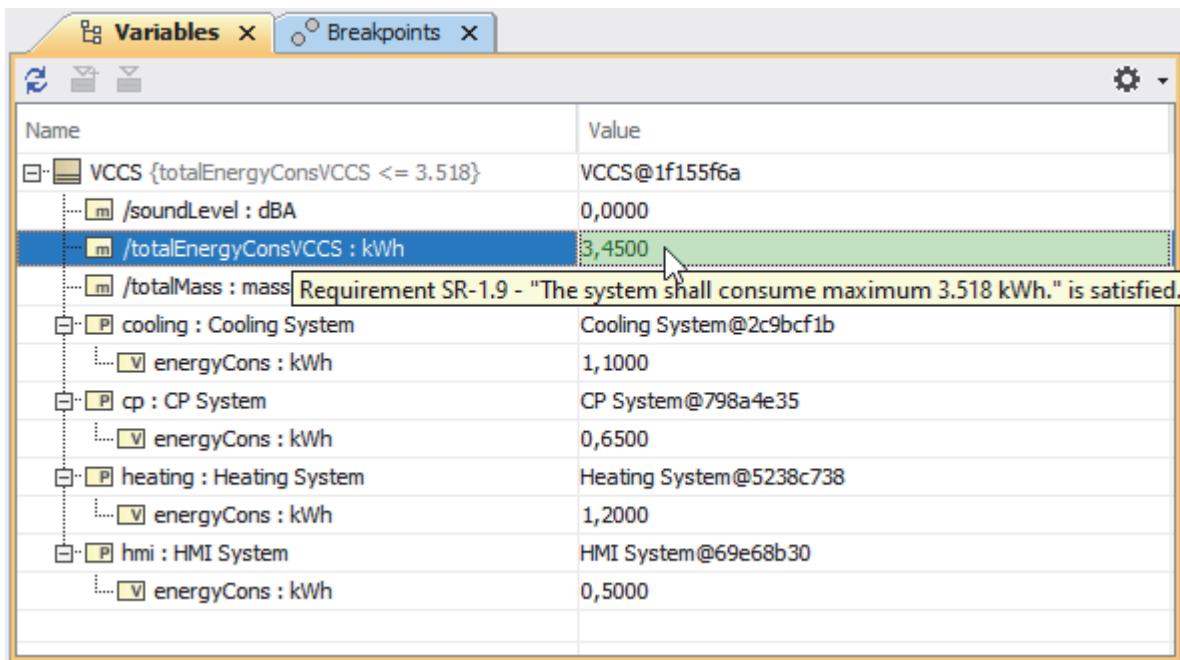


Now, if you hover the mouse over the underlined word *maximum* on the shape of the *Max Energy Consumption* system requirement, you can see the complete condition that must be fulfilled to satisfy that system requirement.



If you go back to the *Total Energy Consumption of VCCS* diagram and execute the model again, you should note that the *totalEnergyConsVCCS* value property is highlighted in green, indicating that the *Max Energy Consumption* system requirement is satisfied. Try to change the input values and see how the highlight

color changes when the requirement is not satisfied. This is how you perform the automated requirements verification. Moreover, this way you can perform manual or automatic (it requires an algorithm to be defined) parameters optimization in order to find the optimal system configuration.



The screenshot shows the 'Variables' tab of the 'Simulation' panel. A requirement message is displayed below the table: 'Requirement SR-1.9 - "The system shall consume maximum 3.518 kWh." is satisfied.' The table lists system and subsystem parameters with their values:

Name	Value
VCCS {totalEnergyConsVCCS <= 3.518}	VCCS@1f155f6a
/soundLevel : dBA	0,0000
/totalEnergyConsVCCS : kWh	3,4500
/totalMass : mass	Requirement SR-1.9 - "The system shall consume maximum 3.518 kWh." is satisfied.
cooling : Cooling System	Cooling System@2c9bcf1b
energyCons : kWh	1,1000
cp : CP System	CP System@798a4e35
energyCons : kWh	0,6500
heating : Heating System	Heating System@5238c738
energyCons : kWh	1,2000
hmi : HMI System	HMI System@69e68b30
energyCons : kWh	0,5000

Step 7. Storing values in the model

System or subsystem parameter values you specify in the **Variables** tab of the **Simulation** panel, as well as the calculated MoE value you see displayed there, can be stored in your model. SysML instance specifications and their slot values can be used for this. Once you have the combination of inputs and the output you would like to keep, you can save them as slot values of the instance of the relevant system architecture blocks (this time, the blocks of the VCCS and its subsystems). Keep in mind that each time you save the instance specifications of these blocks, you are capturing a snapshot of the system at run-time. You can take as many snapshots as you want at different points in time.

For keeping your model well organized, it is recommended to store the instance specifications and their slots in a separate package. They can be displayed in a special type of table: the instance table. It provides a convenient layout for reviewing slot values and recalculating them. Moreover, new instances with corresponding slots can be created here.

To create an instance of the *VCCS* block

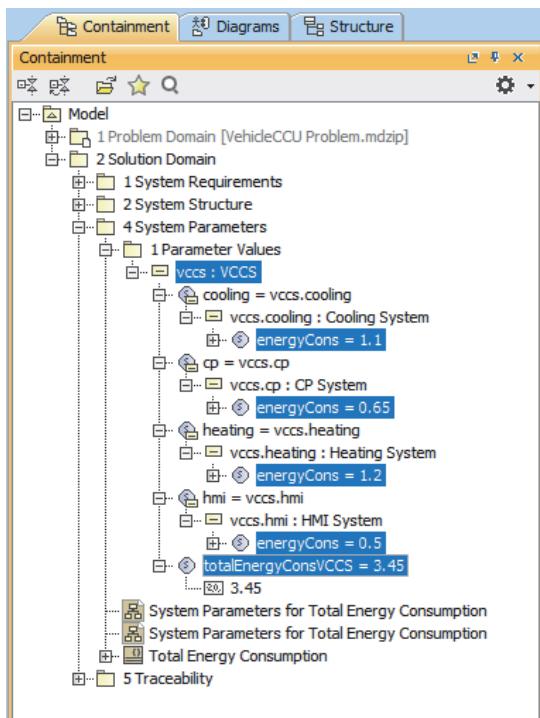
1. In the **Variables** tab of the **Simulation** panel, select the root element; that is, the **VCCS** block.

Name	Value
-> VCCS	VCCS@74dbebe09
m /soundLevel : dB	0,0000
m /totalEnergyConsVCCS : kWh	3,4500
m /totalMass : mass[kilogram]	0,0000
-> P cooling : Cooling System	Cooling System@154b0d41
L v energyCons : kWh	1,1000
-> P cp : CP System	CP System@245b6c66
L v energyCons : kWh	0,6500
-> P heating : Heating System	Heating System@32b035be
L v energyCons : kWh	1,2000
-> P hmi : HMI System	HMI System@25184696
L v energyCons : kWh	0,5000

2. Click the Export to New Instance button  in the toolbar above.
 3. In the **Select Owner** dialog, select the *4 System Parameters* package (you may need to expand the *2 Solution Domain* package beforehand).
 4. Click the **Create Owner** button at the bottom of the dialog and then select **Package**.

i If you don't see the **Create Owner** button, click to toggle the **Creation Mode** button. As a result, the **Create Owner** button appears in the dialog.

5. In the Specification of the new package, type *1 Parameter Values* to give it a name.
 6. Click **Close** in the Specification of that package.
 7. Click **OK** in the **Select Owner** dialog. The instance of the *VCCS* block with slots holding concrete values of relevant value properties is created.

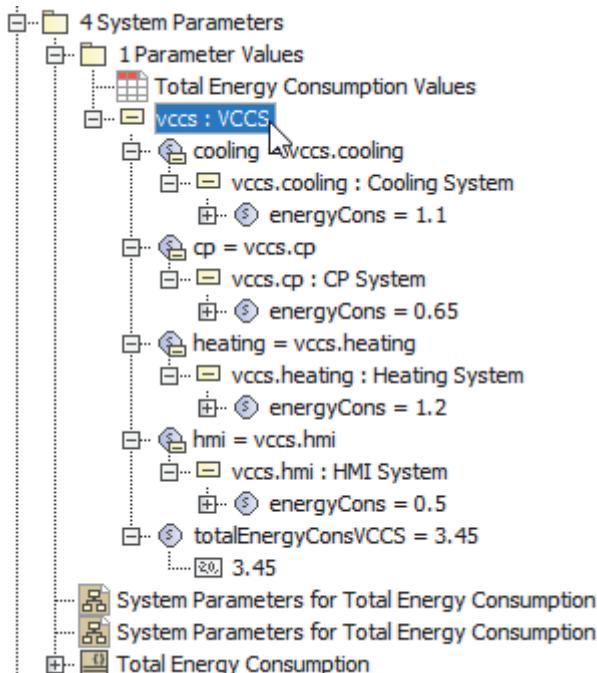


As you can see in the preceding figure, the Model Browser is not the best place for reviewing instances, especially when you have more than a few. In that case, the infrastructure of the instance table should be utilized.

To display the instance of the *VCCS* block in the instance table

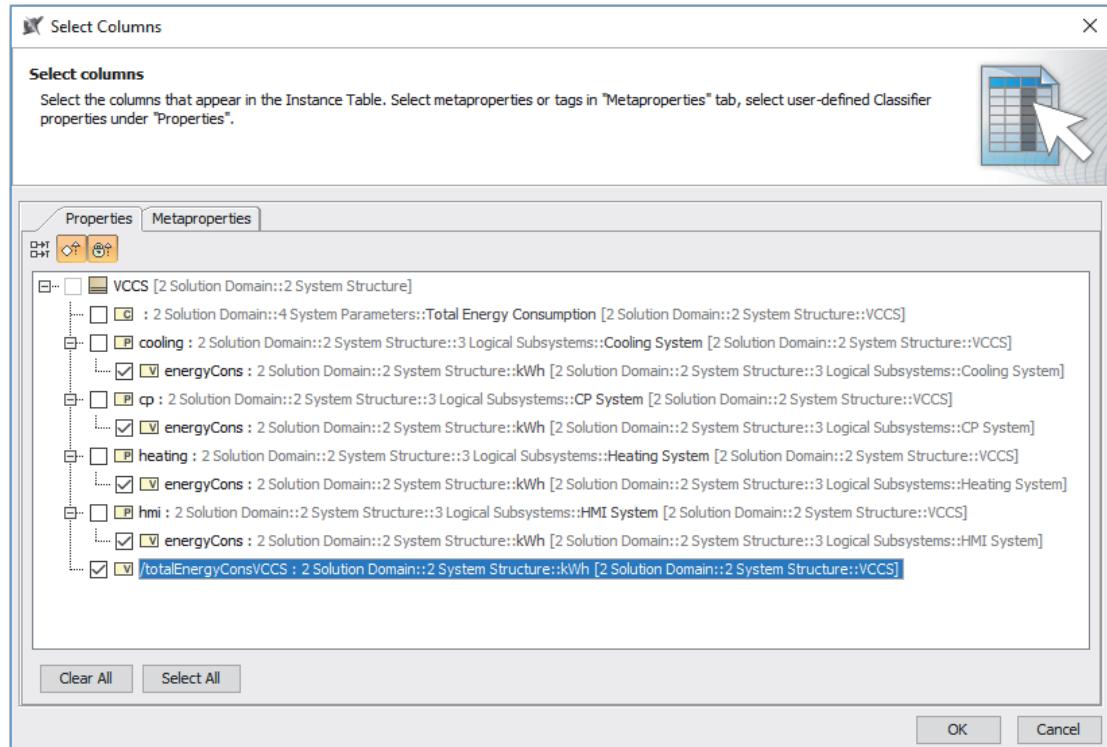
1. Create the instance table:
 - a. In the Model Browser, right-click the *1 Parameter Values* package and select **Create Diagram**.
 - b. In the search box, type *ins*. When you see the instance table selected in the search results, press Enter. The table is created.
 - c. Type *Total Energy Consumption Values* to specify the name of the new table and press Enter again.

2. In the Model Browser, under the same package, select the root instance; that is, the one typed by the *VCCS* block.



3. Drag the selected instance to the instance table on the right. As a result:
 - a. The instance of the *VCCS* block is displayed in the table.
 - b. The *VCCS* block, as it types the selected instance, is automatically set as a classifier of the instance table. You can see it in the **Criteria** area above the table.
4. Hide all the columns except the **Name**:
 - a. Right-click any column.
 - b. Select **Hide**.
 - c. Repeat steps a and b to hide other unneeded columns.
5. Display the columns to show relevant slot values:
 - a. On the toolbar above the instance table, click the **Columns** button and select **Select Columns**.
 - b. In the open dialog, click to select:

- i. All the *energyCons* value properties (you may need to expand the appropriate part properties beforehand). These are the slots' input values.
- ii. The *totalEnergyConsVCCS* value property. This is the slot for the output value.



- c. Click **OK** to close the dialog.

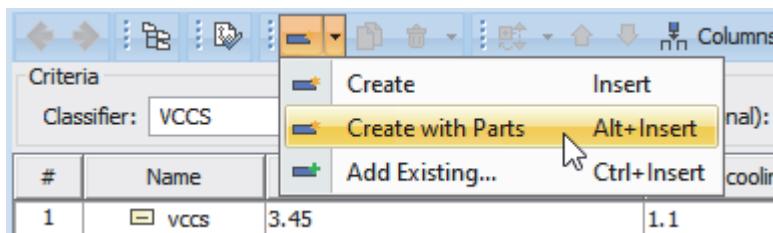
When you're done, the instance table in your model should look the same as the one in the following figure.

#	Name	totalEnergyConsVCCS : kWh	cooling.energyCons : kWh	cp.energyCons : kWh	heating.energyCons : kWh	hmi.energyCons : kWh
1	vccs	3.45	1.1	0.65	1.2	0.5

Now you can create new rows to display other instances with slots for diverse input and output values.

To create a new instance specification of the *VCCS* block in the instance table

1. On the toolbar above the instance table, click the Create (Insert) button and then select **Create with Parts** (see the following figure). This is necessary because you also need to create slots of value properties that belong to the parts of the *VCCS* block, but not to this block directly. A row of the new instance with all slot values 0 is created.



2. In the newly created row, specify the input values.

3. When you're done, on the toolbar above the instance table, click the Run button  and then select **Evaluate Selected Rows**. The result is calculated.

i Note that output values violating the *Max Energy Consumption* system requirement are highlighted. If you cannot see the highlights in your model, make sure the *SimulationProfile.mdzip* is used in your model. For more information on how to use another project or profile in your project, refer to the documentation of the [modeling tool](#).

#	Name	<input checked="" type="checkbox"/> totalEnergyConsVCCS : kWh	<input checked="" type="checkbox"/> cooling.energyCons : kWh	<input checked="" type="checkbox"/> cp.energyCons : kWh	<input checked="" type="checkbox"/> heating.energyCons : kWh	<input checked="" type="checkbox"/> hmi.energyCons : kWh
1	vccs	3.45	1.1	0.65	1.2	0.5
2	vccs1	4.01	1.3	0.78	1.2	0.73

The contents of the instance table can be exported to the MS Excel spreadsheet to share it internally or externally.

System Parameters done. What's next?

With this cell, the task of building the LSA model of the [Sol](#) is almost finished. All that is left to do is establish traceability relationships from the LSA model to system requirements specifications (see Chapter [Traceability to System Requirements](#)).

Traceability to System Requirements

What is it?

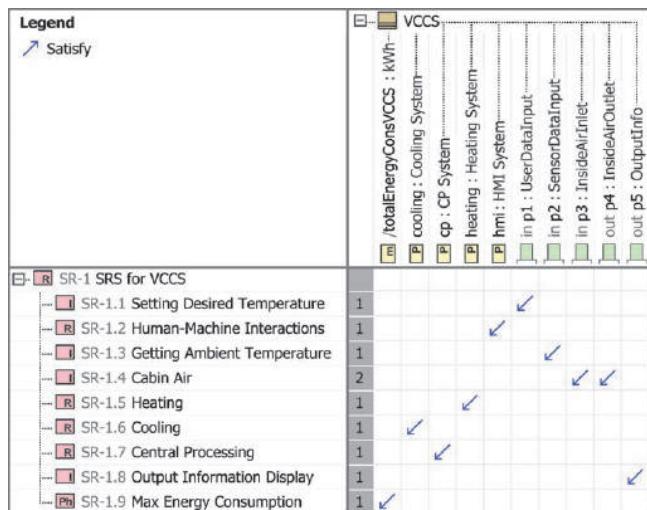
To have a complete LSA model, you need to establish traceability relationships from the elements that capture the LSA of the **Sol** to the elements that capture the system requirements. As determined by SysML, these are satisfy relationships, meaning that the element from the LSA model fulfills one or more of the system requirements. For example, a satisfy relationship can be established from the part property typed by the block that captures some logical subsystem to a certain system requirement. In the complete LSA model, all system requirements are satisfied by one or more elements of the logical system architecture.

Who is responsible?

Traceability relationships in the LSA model can be established by the Systems Architect or Systems Engineer. He/she must be the head of the whole systems engineering project, the one, who has "a big picture view" of the **Sol** and is responsible for the smooth integration of the subsystems and their components into the whole.

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 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.



Tutorial

- Step 1. Creating a matrix for capturing satisfy relationships
- Step 2. Capturing satisfy relationships

Step 1. Creating a matrix for capturing satisfy relationships

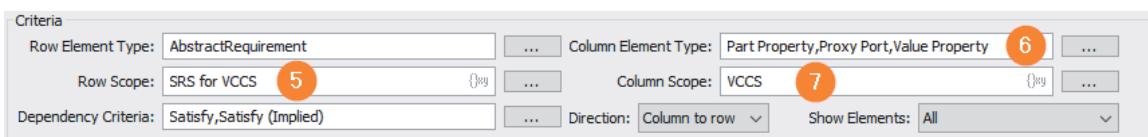
In this step, we create a Satisfy Requirement Matrix and specify its criteria. System requirements can be displayed in the rows of the matrix, and the elements that capture the LSA of the **Sol** can be displayed in its columns.

To create a Satisfy Requirement Matrix

1. In the Model Browser, right-click the *5 Traceability* package (you created it in Chapter [System Requirements](#)) and select **Create Diagram**.
2. In the search box, type *srm*, the acronym of the predefined Satisfy Requirement Matrix, and press Enter.

(i) If you don't see any results, click the **Expert** button below the search results list. The list of available diagrams expands in the Expert mode.
3. Type *LSA to System Requirements* 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. Select the *SR-1 SRS for VCCS* requirement (for this you might need to expand the *1 System Requirements* package first) and drag it onto the **Row Scope** box in the **Criteria** area above the matrix contents. The *SR-1 SRS for VCCS* requirement becomes the scope of the matrix rows (see the following figure).
6. Select column element types:
 - a. In the Model Browser, select any part property, proxy port, and value property.

(i) 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.
 - b. Drag the selection onto the **Column Type** box in the **Criteria** area above the matrix contents. The columns of the matrix are set to display all the part properties, proxy ports, and value properties (see the following figure) within the scope you select in the next step.
7. Select column element scope:
 - a. In the Model Browser, select the *VCCS* block (you might need to expand the *1 System Structure* package beforehand).
 - b. Drag the selection onto the **Column Scope** box in the **Criteria** area above the matrix contents. The selected block becomes the scope of the matrix rows (see the following figure).



Once you're done with the criteria, the contents of the matrix is updated. All the cells are empty at the moment, except the one at the intersection of the `totalEnergyConsVCCS` value property and the `SR-1.9 Max Energy Consumption` requirement, if you have created the satisfy relationship in [step 6 of the System Parameters cell tutorial](#).

Legend		VCCS								
		/totalEnergyConsVCCS : kWh								
		cooling : Cooling System			heating : Heating System			hmi : HMI System		
		in	out	cp	in	out	cp	in	out	in
		p1	p2	p3	p4	p5	p6	p7	p8	p9
SR-1 SRS for VCCS										
R SR-1.1 Setting Desired Temperature R SR-1.2 Human-Machine Interactions R SR-1.3 Getting Ambient Temperature R SR-1.4 Cabin Air R SR-1.5 Heating R SR-1.6 Cooling R SR-1.7 Central Processing R SR-1.8 Output Information Display Ph SR-1.9 Max Energy Consumption										
		1								

Step 2. Capturing satisfy relationships

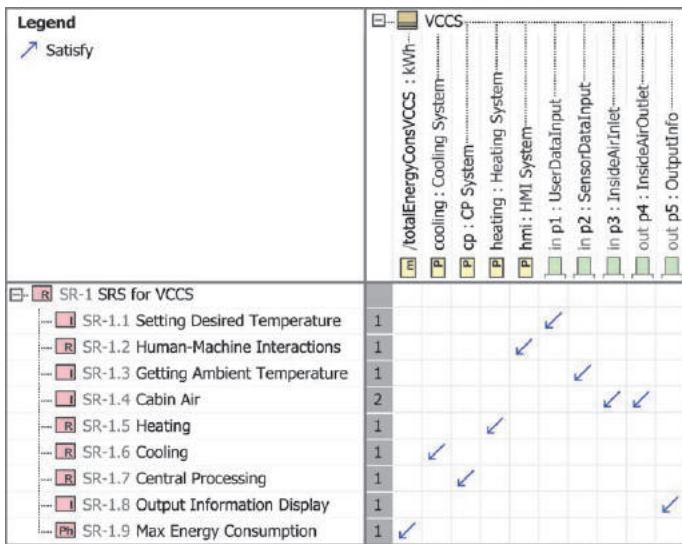
Now you're ready to specify other satisfy relationships. In the matrix, this can be done very easily.

To specify a satisfy relationship from the `p1 : UserDataInput` proxy port to the `SR-1.1 Setting Desired Temperature` requirement

- Double-click the cell at the intersection of the column that displays the proxy port and the row that displays the requirement. The satisfy relationship is created between the appropriate items and displayed in the cell.

Legend		VCCS								
		/totalEnergyConsVCCS : kWh								
		cooling : Cooling System			heating : Heating System			hmi : HMI System		
		in	out	cp	in	out	cp	in	out	in
		p1	p2	p3	p4	p5	p6	p7	p8	p9
SR-1 SRS for VCCS										
R SR-1.1 Setting Desired Temperature R SR-1.2 Human-Machine Interactions R SR-1.3 Getting Ambient Temperature R SR-1.4 Cabin Air R SR-1.5 Heating R SR-1.6 Cooling R SR-1.7 Central Processing R SR-1.8 Output Information Display Ph SR-1.9 Max Energy Consumption		1								
		1								

We assume that you continue filling in the matrix yourself. If you need a hint for which elements can be related, look at the following figure.



LSA done. What's next?

Once you're done with the LSA, you have the following:

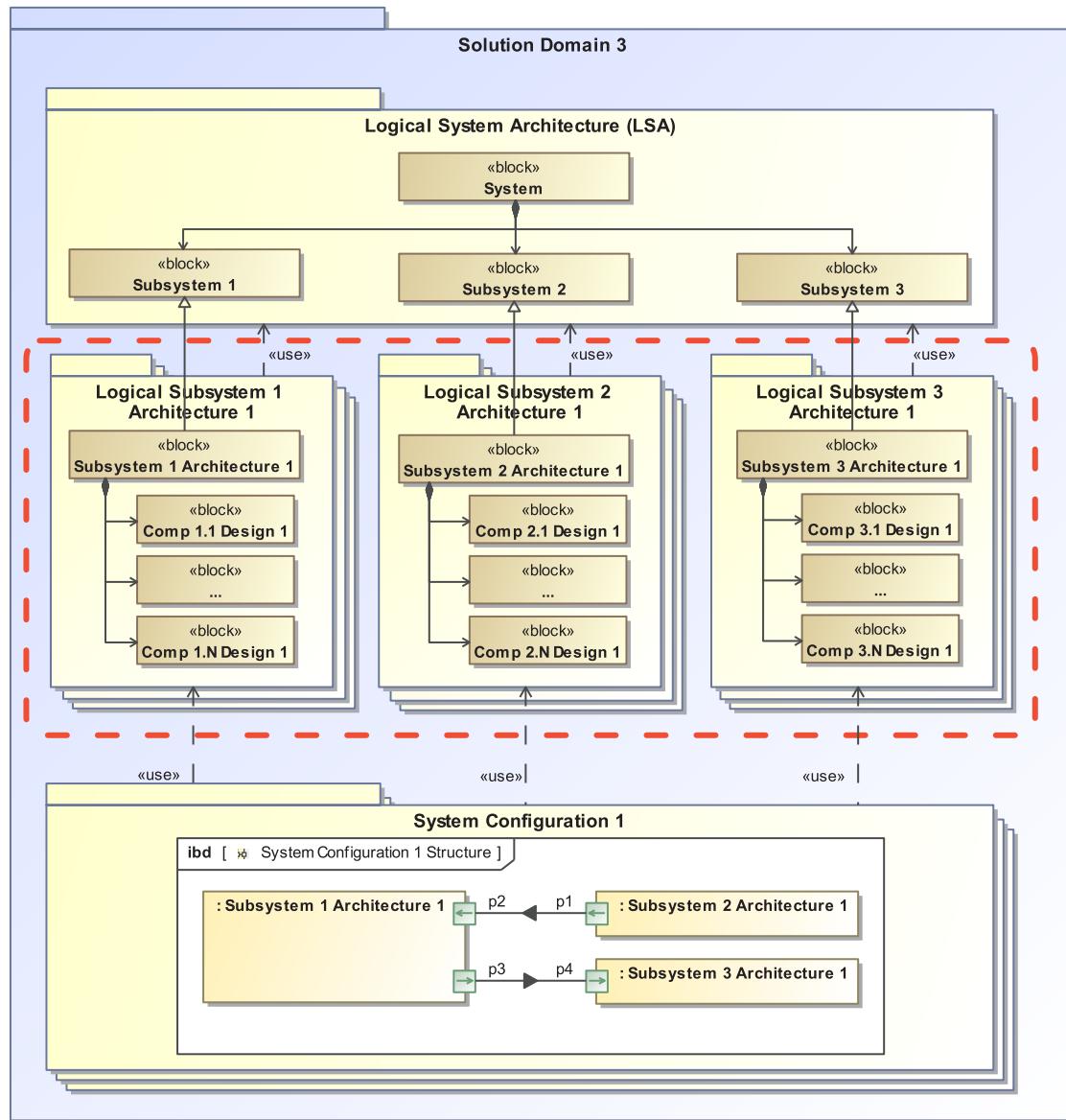
- Logical subsystems
- Logical interfaces
- Exchange items

Having this information, you are ready to specify the solution architecture of each logical subsystem.

Building the logical architecture of subsystems

Once the systems architect identifies the logical subsystems of the **Sol**, he/she can distribute them to separate engineering teams to begin working on logical subsystem architectures (LSSA). Therefore, this phase of creating the solution domain model is devoted to building the architectures of all logical subsystems of the **Sol**. In this phase you should imagine being a member of the engineering team working on the LSSA of a single logical subsystem of the VCCS.

As noted previously, an LSSA of each logical subsystem should be stored in a separate model, apart from the problem domain definition and LSA model. The highlighted area in the following figure shows the part of the solution domain we are talking about. As you can see, these models can be developed in parallel.



A complete LSSA model consists of the specification of subsystem requirements, structure, behavior, and parameters. Additionally, every SysML element you create in this model must satisfy at least one subsystem requirement; otherwise, there is no motivation for having it in the LSSA model. Therefore, you have to start and end the LSSA model with subsystem requirements: first specify them and, when your model is complete, establish traceability relationships between them and SysML elements.

	Pillar						
Domain			Requirements	Structure	Behavior	Parameters	Safety & Reliability
	Problem	Black Box	Stakeholder Needs	System Context	Use Cases	Measures of Effectiveness (MoEs)	Conceptual and Functional Failure Mode & Effects Analysis (FMEA)
		White Box		Conceptual Subsystems	Functional Analysis	MoEs for Subsystems	Conceptual Subsystems FMEA
	Solution	System Requirements	System Structure	System Behavior	System Parameters	System Safety & Reliability (S&R)	
		Subsystem Requirements	Subsystem Structure	Subsystem Behavior	Subsystem Parameters	Subsystem S&R	
		Component Requirements	Component Structure	Component Behavior	Component Parameters	Component S&R	
	Implementation	Implementation Requirements					

A logical subsystem in this phase of building the solution domain model can be referred to as the subsystem of interest. Moreover, from the point of view of the appointed engineering team, it is referred to as the system of interest.

Note that the following material mainly focuses on building the logical architecture of a single subsystem of interest. This is the Cooling System of the VCCS. You should keep in mind that while you build the logical architecture for the Cooling System, other engineers / engineering teams will work in parallel to build the logical architectures of the CP System, Heating System, and HMI System.

Subsystem Requirements

What is it?

To provide the solution architecture of the logical subsystem, the responsible engineering team needs to analyze the relevant subsystem requirements specification, which is produced when having the complete LSA model. Each subsystem requirement from the leaf level of the specification, should point either to a particular system requirement or an element from the LSA model (e.g., value property or proxy port of the block which represents the relevant logical subsystem). This proves the necessity of the subsystem requirement (just like system requirements point to stakeholder needs or problem domain model elements).

After you are done with the LSSA, you should establish satisfy relationships from the LSSA elements to subsystem requirements in order to assert which LSSA elements fulfill which subsystem requirements. This is described in Chapter [Traceability to Subsystem Requirements](#).

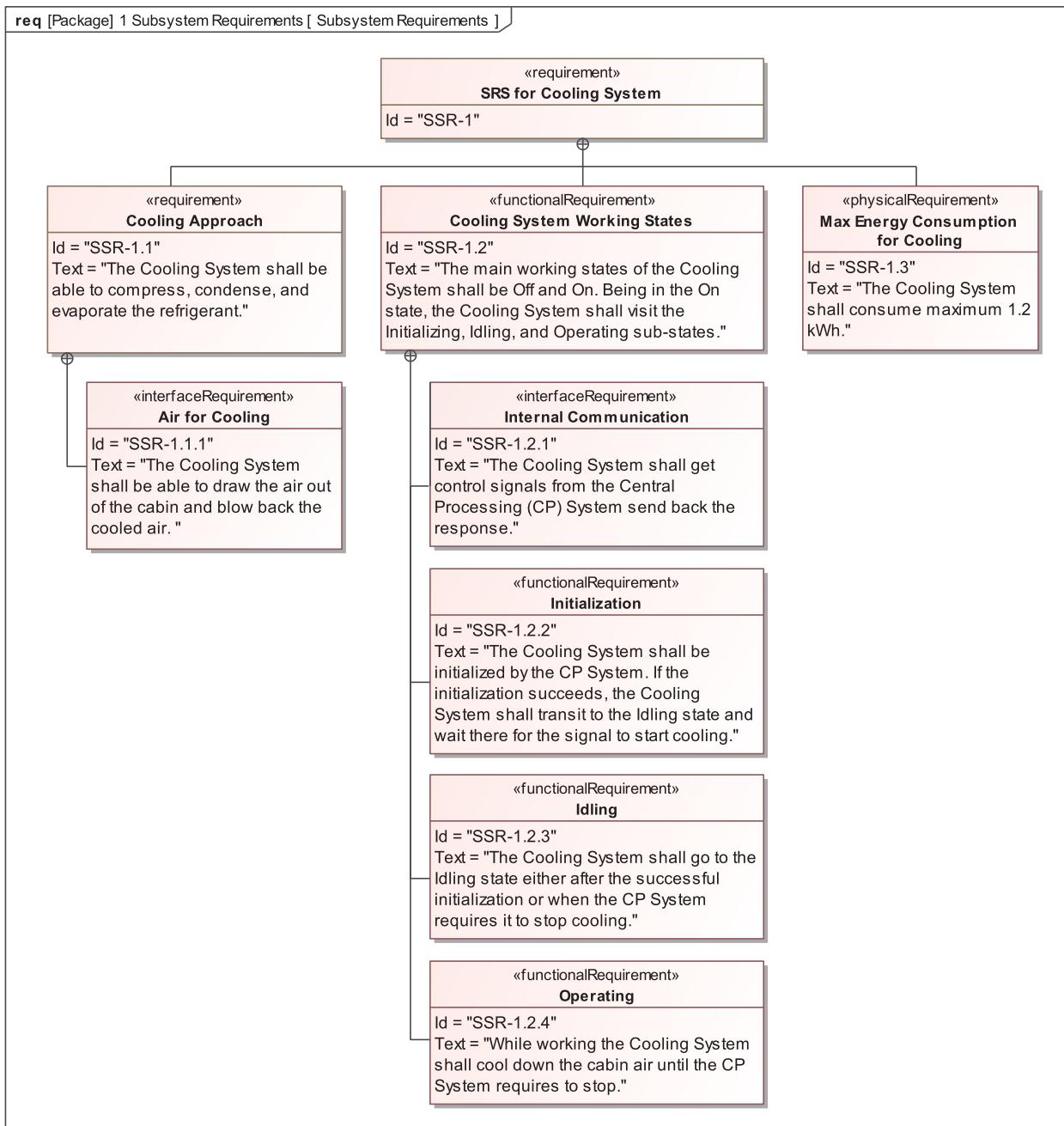
Who is responsible?

The subsystem requirements specification can be produced by the Systems Architect or Systems Engineer, who has built the LSA model.

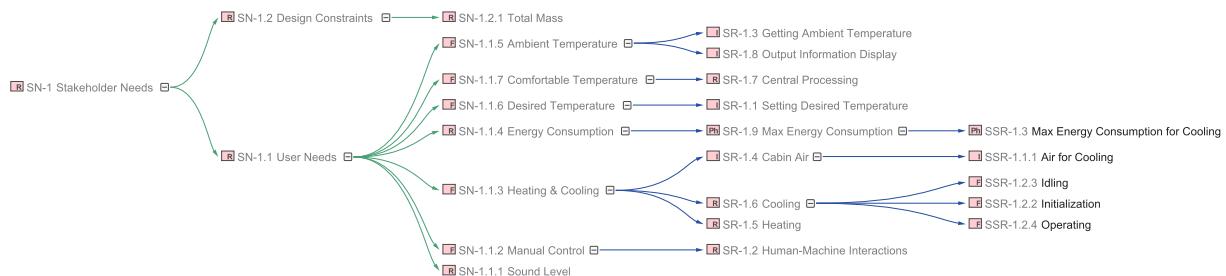
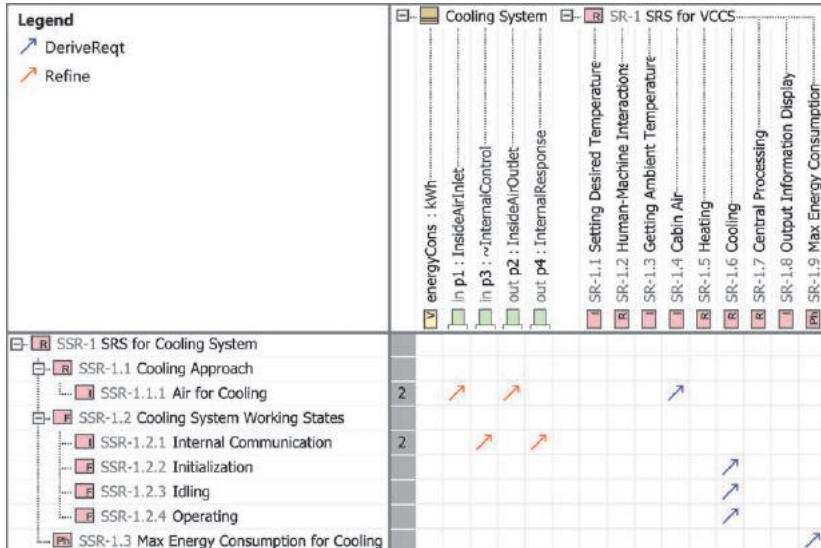
How to model?

In the **modeling tool**, a single subsystem requirement can be captured as the SysML requirement. Just like a system requirement, the requirement for a subsystem must have a unique identification, name, and textual specification, which must be formal and written by following certain guidelines (examples of such guidelines are provided in Chapter [System Requirements](#)).

As with system requirements, subsystem requirements can be categorized and organized into hierarchies. A SysML requirement diagram or table can be used to capture and display them.



Matrices and maps can be used to establish and review the cross-cutting relationships between the subsystem requirements and system requirements or LSA model elements.

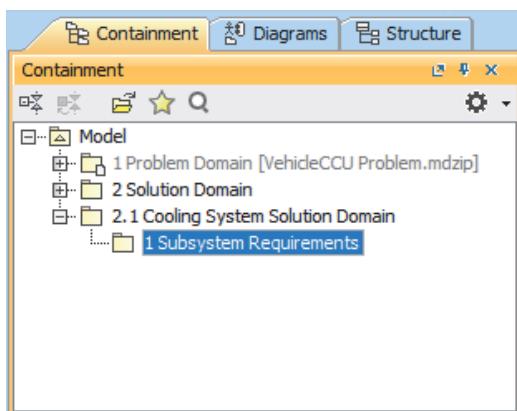


Tutorial

- Step 1. Creating and organizing a model for subsystem requirements
- Step 2. Creating a diagram for subsystem requirements
- Step 3. Specifying subsystem requirements
- Step 4. Specifying traceability relationships

Step 1. Creating and organizing a model for subsystem requirements

To begin specifying subsystem requirements, you need to establish an appropriate structure of the model. Following the layout of the MagicGrid framework, model artifacts that capture subsystem requirements should be stored under the structure of packages, as displayed in the figure below. The upper-level package represents the domain, and the lower-level package stands for the cell.



However, this is not exactly what we are going to create. In real-world projects, the *2.1 Cooling System Solution Domain* package together with its subpackage (and all subsequently added ones) would appear in another model separate from the LSA model. This is necessary because the engineering team appointed to build the LSSA model should only be able to read it, and not make any changes in the LSA model.

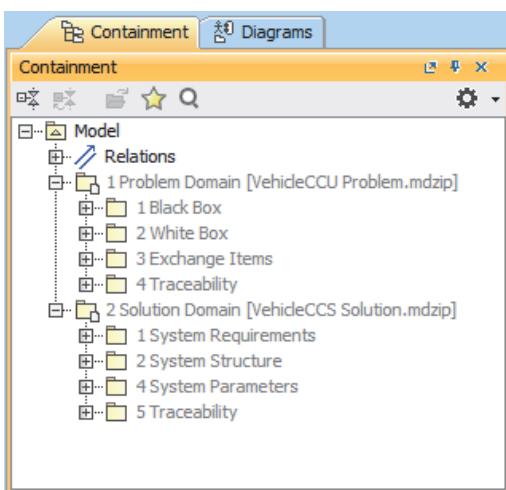
Therefore, before organizing packages for building the LSSA of the Cooling System, you need to create a new model and then, in terms of the **modeling tool**, use the LSA model in it. It should be noted here, that an LSSA model should be created for each logical subsystem identified in the LSA.

To create a new model for the LSSA of the Cooling System and organize it to capture the subsystem requirements specification

1. Start sharing the LSA model. Once you share it, the problem domain analysis model used in the LSA model becomes shared, too.

i 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. This is the solution architecture model of the Cooling System, so it can be named *Cooling System Solution.mdzip*.
3. Use the LSA model as *read-only* in the LSSA model of the Cooling System. The problem domain analysis model is automatically used as well.



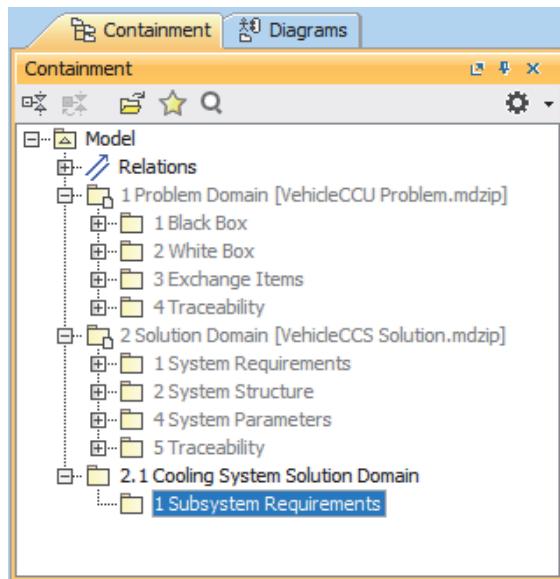
i Names in grey indicate elements that cannot be modified, as the problem domain model is used in the solution domain model as *read-only*.

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.

i Let's agree that packages for storing the solution architectures of other logical subsystems (such as the CP System and Heating System, among others) 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. In the search box, type *pa*, the first two letters of the element type *Package*, and press Enter.

9. Type *1 Subsystem Requirements* to specify the name of the new package and press Enter.



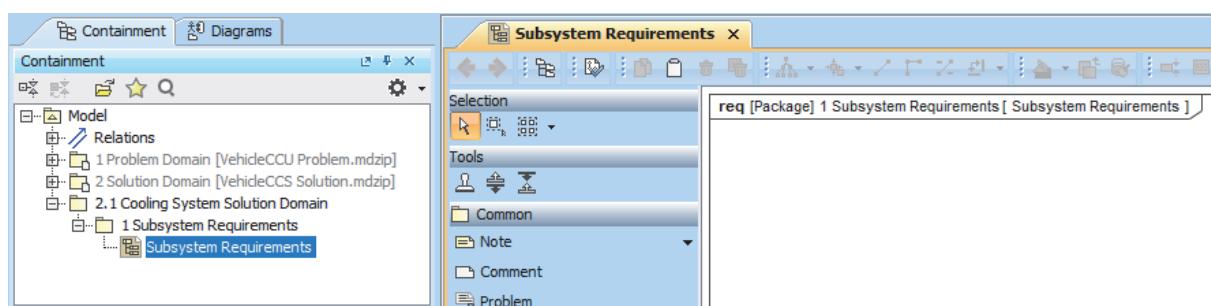
Step 2. Creating a diagram for subsystem requirements

For capturing and displaying subsystem requirements, we recommend using the SysML requirement diagram or table. Let's create a diagram this time.

To create the SysML requirement diagram for specifying subsystem requirements

1. Right-click the *1 Subsystem Requirements* package and select **Create Diagram**.
2. In the search box, type *rd*, where *r* stands for *requirement* and *d* for *diagram*, and then double-press Enter. The diagram is created.

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



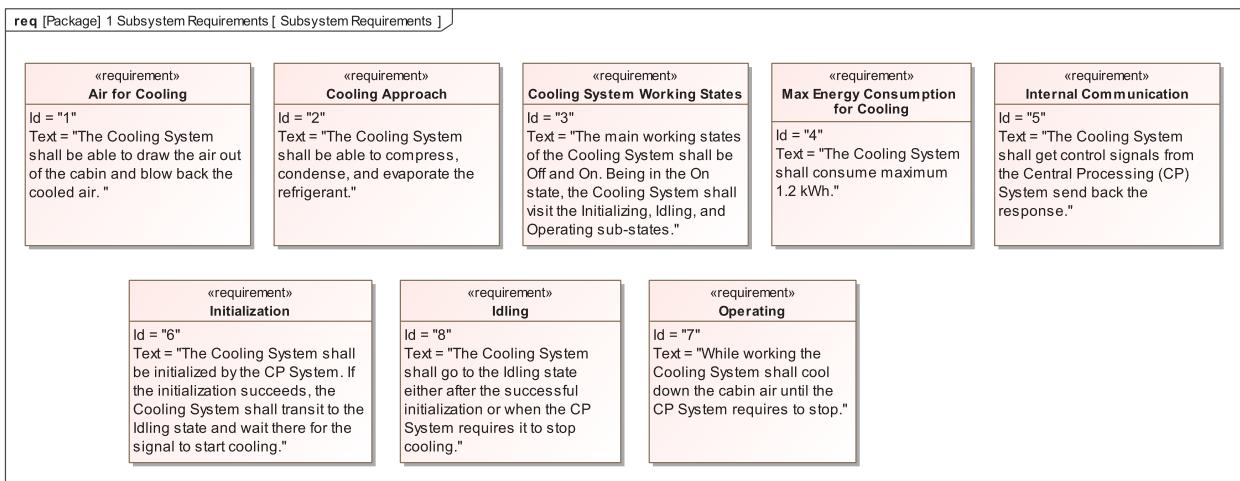
Step 3. Specifying subsystem requirements

Let's say that the engineering team working on the solution architecture of the Cooling System receives from the System Architect the following subsystem requirements:

#	Name	Text
1	Air for Cooling	The Cooling System shall be able to draw the air out of the cabin and blow back the cooled air.

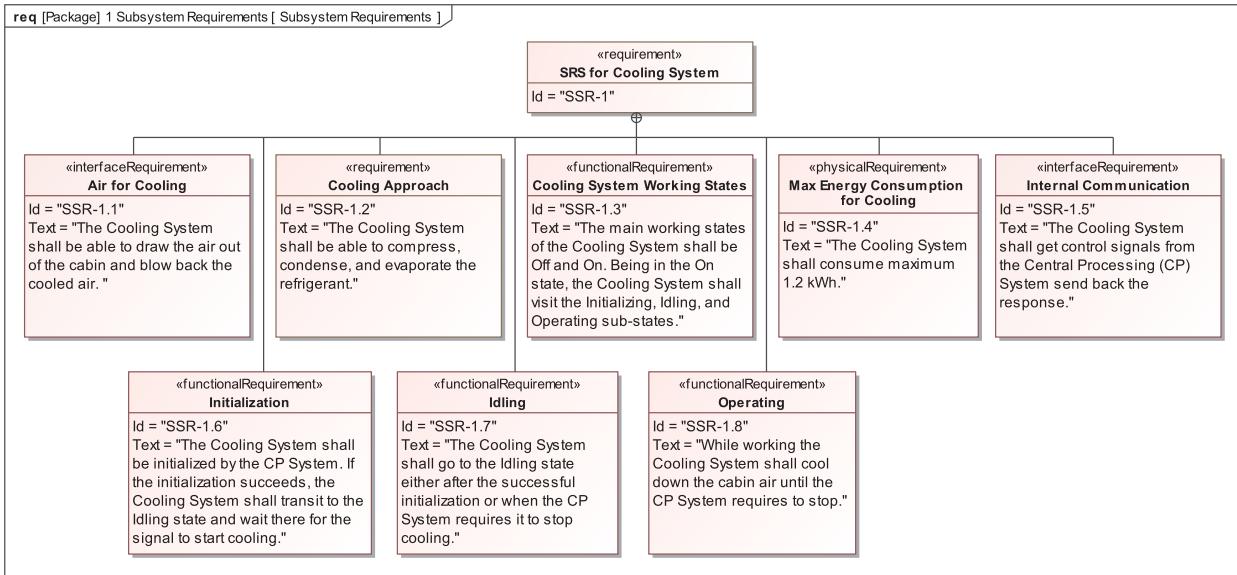
#	Name	Text
2	Cooling Approach	The Cooling System shall be able to compress, condense, and evaporate the refrigerant.
3	Cooling System Working States	The main working states of the Cooling System shall be Off and On. Being in the On state, the Cooling System shall visit the Initializing, Idling, and Operating sub-states.
4	Max Energy Consumption for Cooling	The Cooling System shall consume maximum 1.2 kWh.
5	Internal Communication	The Cooling System shall get control signals from the Central Processing (CP) System send back the response.
6	Initialization	The Cooling System shall be initialized by the CP System. If the initialization succeeds, the Cooling System shall transit to the Idling state and wait there for the signal to start cooling.
7	Idling	The Cooling System shall go to the Idling state either after the successful initialization or when the CP System requires it to stop cooling.
8	Operating	While working the Cooling System shall cool down the cabin air until the CP System requires to stop.

As you already know how to capture requirements in the diagram, we assume that you can do this on your own. If you need to refresh your knowledge, refer to the procedure in step 3 of the [System Requirements](#) tutorial. When you are done, the *Subsystem Requirements* diagram in your model should look very similar to the one below.

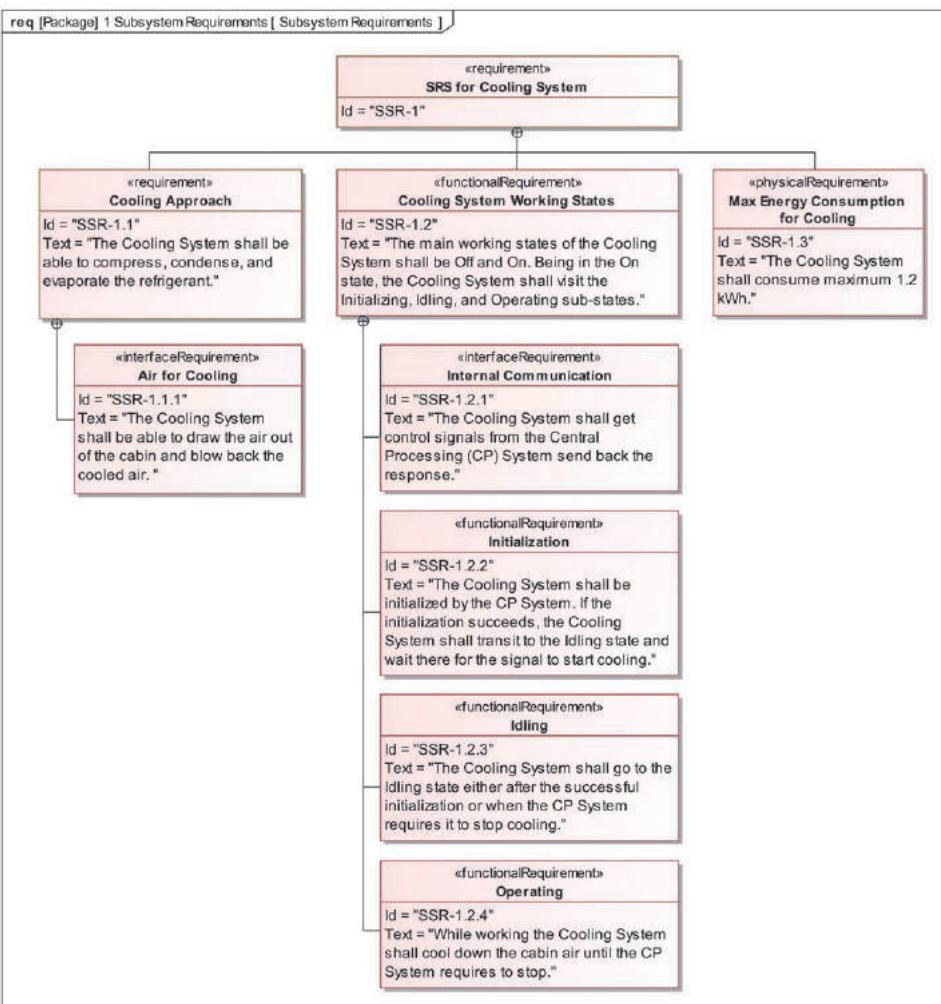


You also need to group, number, and categorize the subsystem requirements. These topics are described in [step 4](#), [step 5](#), and [step 6](#) of the [Stakeholder Needs](#) tutorial, so we assume that you can do this on your own, too. As you can see in the following figure, subsystem requirements should be:

- Grouped under the *SRS for Cooling System* grouping requirement (if you need a reminder on how to display relationships on the diagram, refer to the procedure in [step 3](#) of the [System Requirements](#) tutorial)
- Numbered using the *SSR-* prefix
- Categorized into interface, functional, non-functional, and physical requirements



This flat structure of subsystem requirements can be organized into a deeper hierarchical structure, as shown in the figure below.

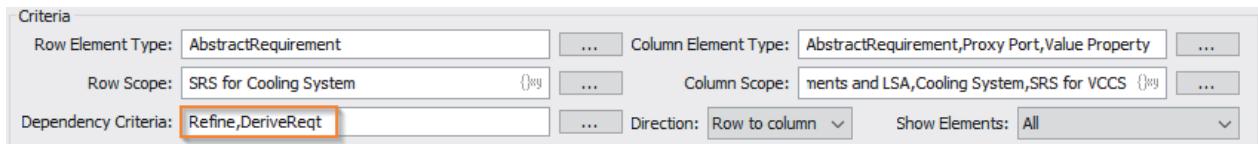


Step 4. Specifying traceability relationships

As mentioned before, the necessity of any subsystem requirement must be proved. That is, the subsystem requirement can be created either to derive a system requirement or refine an LSA model element. In the

SysML model, the deriveReqt relationship is used to assert the derivation, and the refine relationship to assert the refinement. Both types of relationships can be specified either in a SysML requirements diagram or a matrix. We recommend using the matrix for a larger scope, because it provides a more compact view of the model than the diagram does.

In this case, we recommend utilizing the infrastructure of one of the predefined matrices in the [modeling tool](#): either the Derive Requirement Matrix or the Refine Requirement Matrix. Whatever matrix you choose, unlike in other cases, you need to update the dependency criteria to set up the matrix criteria. Hence, if it is the Derive Requirement Matrix, you need to set the refine relationship as dependency criteria in addition to the deriveReqt relationship, and if it is the Refine Requirement Matrix, you need to set the deriveReqt relationship as dependency criteria in addition to the refine relationship. The following figure displays how the dependency criteria in your matrix should look like.



The following procedure shows how to create only one of the matrices, the Derive Requirement Matrix, but you can create the alternative one on your own and see which one suits better for you.

As in other models, traceability views and relationships should be stored in a separate package. Because you already know how to create packages, we assume that you can do this on your own. For keeping your model well-organized according to the MagicGrid framework, that package should always be the last sub-package of the *2.1 Cooling System Solution Domain* package. Therefore, its name must begin with number 5 (for example, *5 Traceability*).

To create a Derive Requirement Matrix

1. In the Model Browser, right-click the *5 Traceability* package and select **Create Diagram**.
2. In the search box, type *drm*, the acronym for the predefined Derive Requirement Matrix, and press Enter.

i 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 *Subsystem Requirements to System Requirements and LSA* to specify the name of the new matrix and press Enter again.
4. In the Model Browser, select the *SSR-1 SRS for Cooling System* requirement and drag it onto the **Row Scope** box in the **Criteria** area above the matrix contents. The *SSR-1 SRS for Cooling System* requirement becomes the scope of the matrix rows.
5. Select the column element types:
 - a. Holding down the Ctrl key, in the Model Browser, select any part property, proxy port, and value property.
 - b. Drag the selection onto the **Column Type** box in the **Criteria** area above the matrix contents. The columns of the matrix are set to display all the part properties, proxy ports, and value properties in addition to requirements, which is set by default, within the scope you select in the next step.
6. Select column element scope:
 - a. In the Model Browser, select the *SR-1 SRS for VCCS* requirement (for this you might need to expand the following packages beforehand: *2 Solution Domain* and *1 System Requirements*).

- b. Press the Ctrl key and holding it down, select the *Cooling System* block (for this you might need to expand the *1 System Structure* package first).
 - c. Drag the selection onto the **Column Scope** box in the **Criteria** area above the matrix contents. The *SR-1 SRS for VCCS* requirement and the *Cooling System* block become the scope of the matrix columns.
7. Set the refine relationship as dependency criteria:

- a. Click the button next to the **Dependency Criteria** box.
- b. On the left side of the open dialog, select **Simple Navigation**.
- c. In the search box at the bottom of the dialog, type *refine*.
- d. Select the **Refine [Abstraction] (SysML)** and click to set the **Is Applied** check box value to *true*.

Simple Navigation			
Relation Criterion	Is Applied	Direction	Style
<input type="checkbox"/> Relations			
Refine [Abstraction](SysML)	<input checked="" type="checkbox"/> true	Source To Target	→
Refine [Abstraction](StandardProfile)	<input type="checkbox"/> false		
<input type="checkbox"/> Properties (Traceability::Other)			
Refines	<input type="checkbox"/> false		
<input type="checkbox"/> Properties (Traceability)			
Refined By («AbstractRequirement»)	<input type="checkbox"/> false		

i It is also worth to change the style of the Refine relationship so it differs from the DeriveReqt relationship in the matrix (see the following figure). For more information, refer to the documentation of the [modeling tool](#).

Relation Criterion	Is Applied	Direction	Style
<input type="checkbox"/> Relations			
Refine [Abstraction](SysML)	<input checked="" type="checkbox"/> true	Source To Target	→
Refine [Abstraction](StandardProfile)	<input type="checkbox"/> false		

- e. Click **OK** to close the dialog. Together with the deriveReqt relationship, the refine relationship is set as dependency criteria for the matrix.
8. Click the **Direction** box and select **Row to Column** to enable the creation of relationships pointing from subsystem requirements, which are row elements, to system requirements of LSA elements, which are column elements.
9. 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.

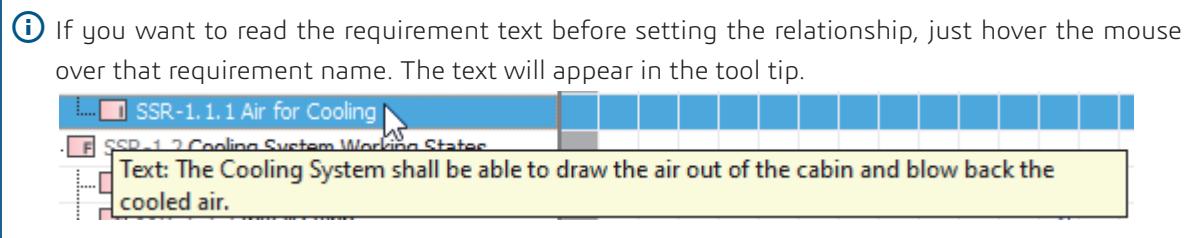
The following figure displays the **Criteria** area of the matrix, with highlights on step 4, 5, 6, 7, and 8 related criteria.

Criteria	Row Element Type: AbstractRequirement	Column Element Type: AbstractRequirement,Part,Property,Proxy Port,V...	5
Row Scope: SRS for Cooling System	4
Dependency Criteria: Refine,DeriveReqt	7
Column Scope: Cooling System,SRS for VCCS	6
Direction: Row to column	8	Show Elements: All	...

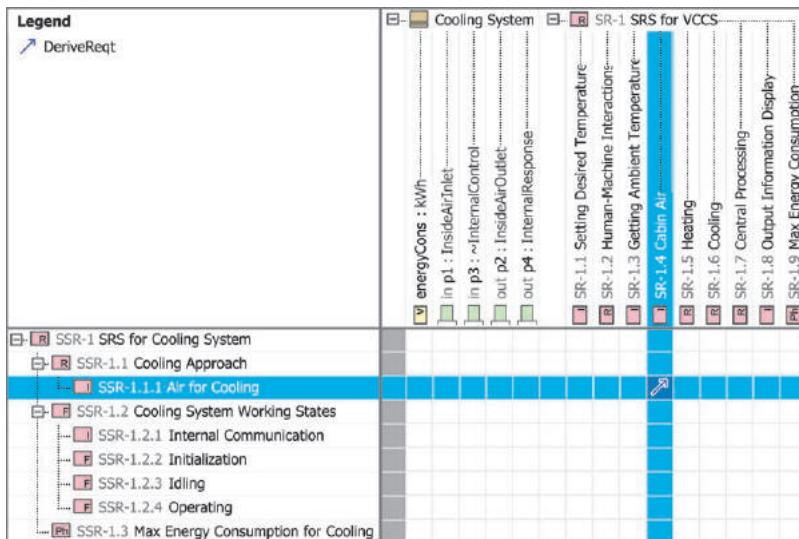
Now you're ready to establish both derivation and refine relationships. Let's create the derivation between *SSR-1.1.1 Air for Cooling* and *SR-1.4 Cabin Air*. This will convey that the *SSR-1.1.1 Air for Cooling* subsystem requirement is created because of the *SR-1.4 Cabin Air* system requirement.

To specify the derivation relationship in the matrix

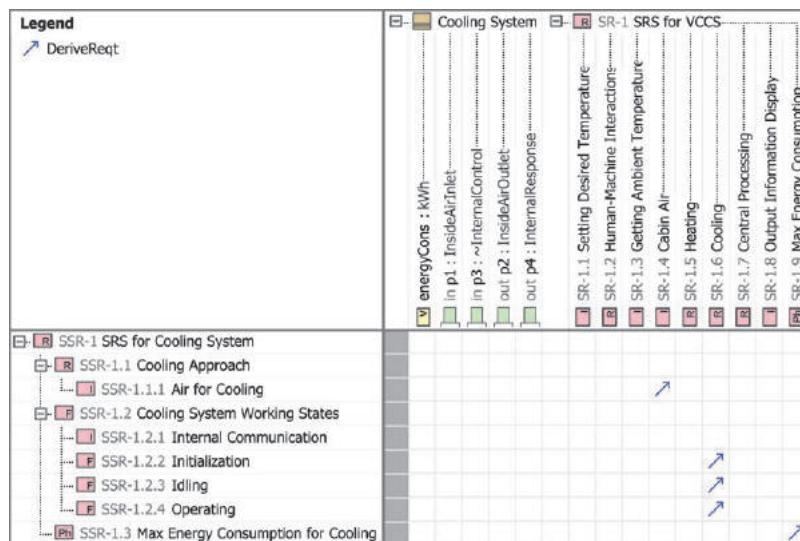
- Double-click the cell at the intersection of the row that displays *SSR-1.1.1 Air for Cooling* and the column that displays *SR-1.4 Cabin Air*.



- Select **DeriveReqt** to create a derivation relationship. The derivation relationship is created between the appropriate items and displayed in the cell.



After all derivation relationships are established, the *Subsystem Requirements to System Requirements and LSA* matrix in your model should resemble the one in the following figure.

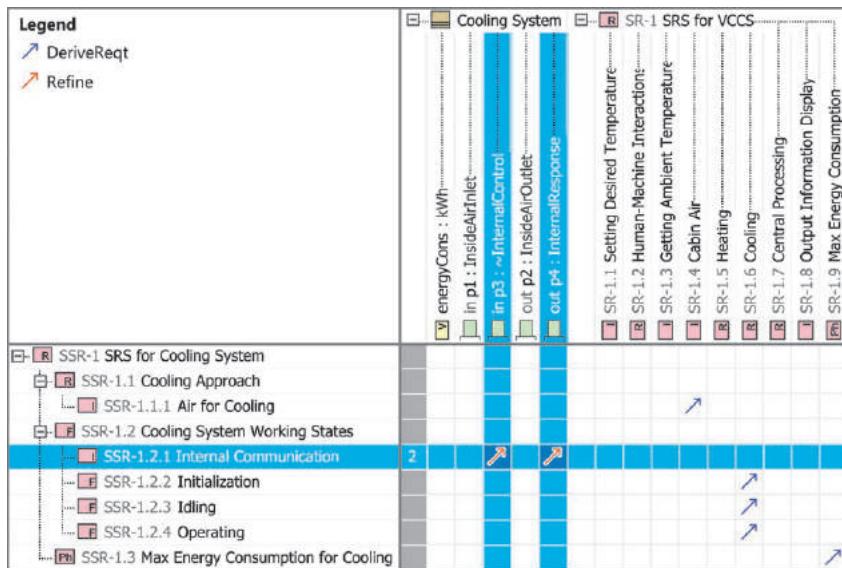


As you can see, the *SSR-1.2.1 Internal Communication* subsystem requirement has no derivation to any system requirement. This is because the *SSR-1.2.1 Internal Communication* subsystem requirement was

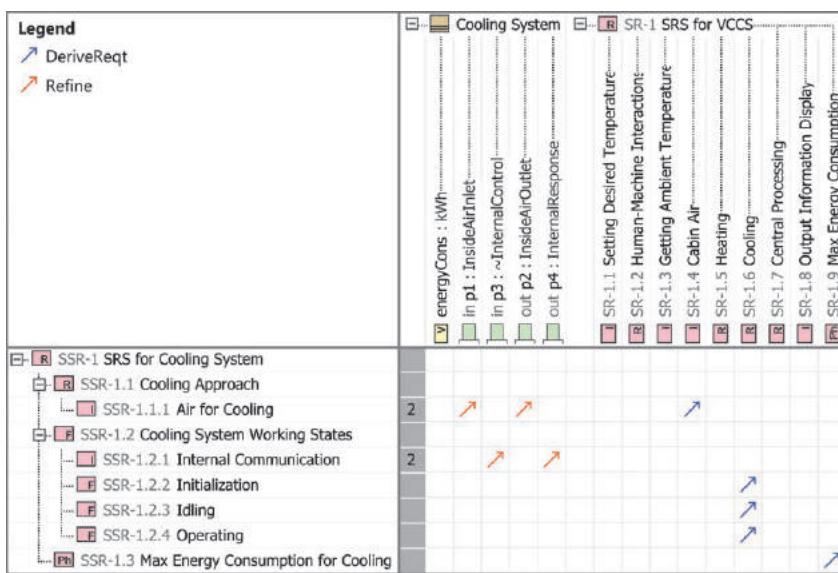
created to refine the *p3* and *p4* proxy ports of the *Cooling System* block. Let's specify appropriate relationships in the matrix.

To specify the refine relationship in the matrix

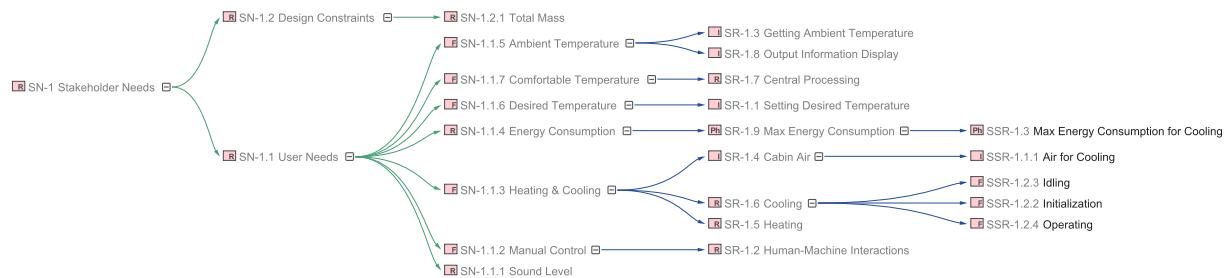
- Double-click the cell at the intersection of the column that displays the *p3* or *p4* proxy port and the row that displays the *SSR-1.2.1 Internal Communication* subsystem requirement. The refine relationship is created between the appropriate items and displayed in the cell.



Although derived from the system requirement, the *SSR-1.1.1 Air for Cooling* subsystem requirement also refines the *p1* and *p2* proxy ports of the *Cooling System* block from the LSA model. We assume, you create appropriate relationships on your own.



Afterwards, a Requirement Derivation Map can be created, with blue arrows representing the derivation relationships between requirements from different domains and green arrows the containment.



Subsystem Requirements done. What's next?

When you have the requirements for a particular logical subsystem of the **Sol**, you can build the logical solution architecture of that subsystem. It begins with specifying the subsystem structure, which means you can move to the [Subsystem Structure](#) cell.

Subsystem Structure

What is it?

You can begin building the solution architecture of a subsystem by defining the internal structure of that subsystem and specifying how its components operate together, and how they interact with other subsystems of the same **Sol**.

As you may recall, the interfaces for interacting with other subsystems are defined in the LSA model by the system architect. These are the constraints the engineering team appointed to build the solution architecture of a single subsystem have to deal with. This ensures the model integrity of the whole **Sol**.

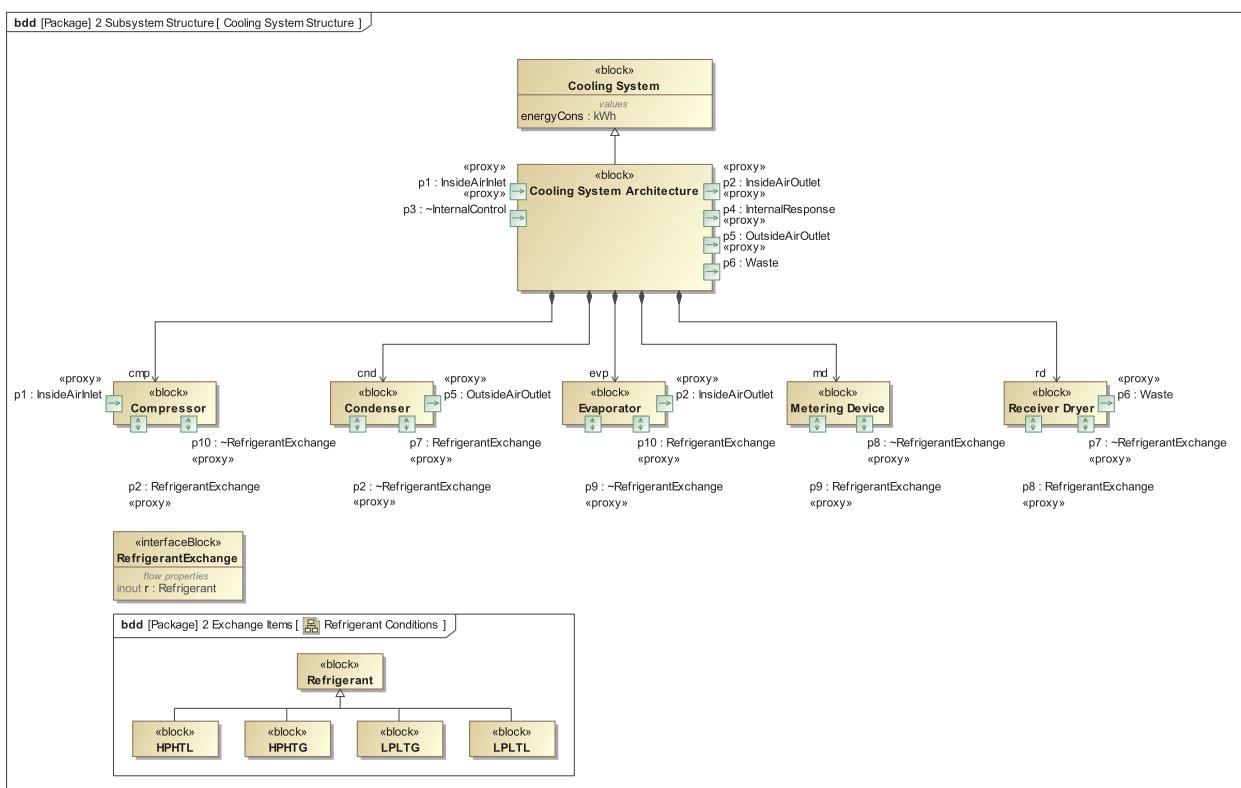
If changes are required (for example, an additional interface is necessary, or an existing interface is acknowledged as not needed) the engineering team applies to the systems architect, who considers the request. If the request is accepted, other engineering teams must be informed about the change to ensure integrity at the system level.

Who is responsible?

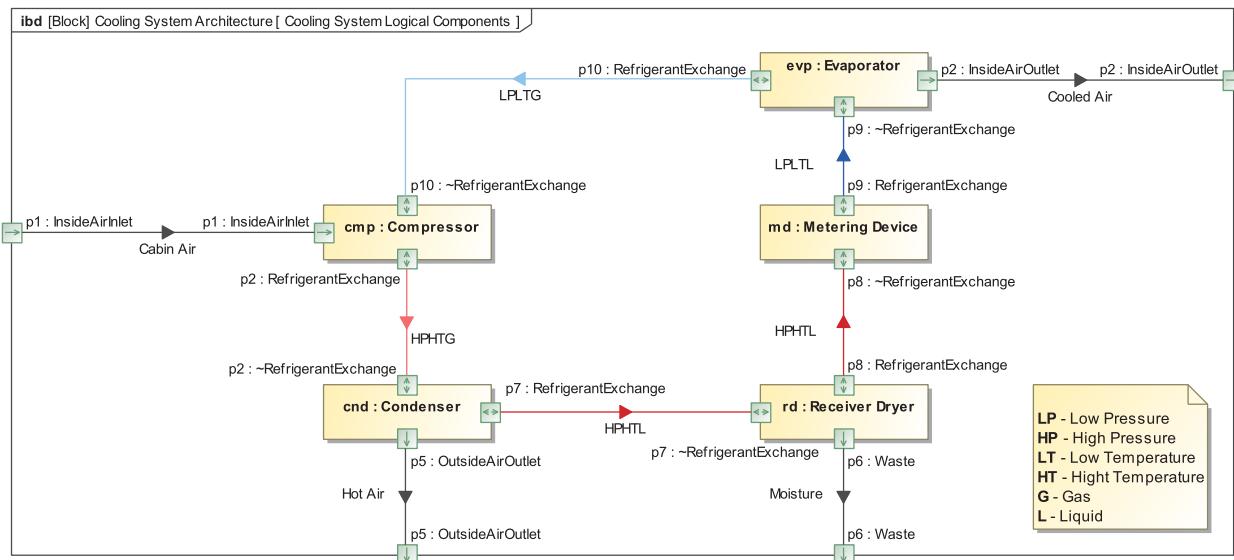
The structure of the appointed logical subsystem can be captured by the Systems Architect, or the Systems Engineer who belongs to the engineering team responsible for building the solution architecture of that logical subsystem.

How to model?

In the **modeling tool**, the components of the subsystem can be defined by using the infrastructure of the SysML block definition diagram (bdd). Components themselves can be captured as blocks, and interfaces as proxy ports typed by appropriate interface blocks.



To specify how the parts of the subsystem operate together and 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.

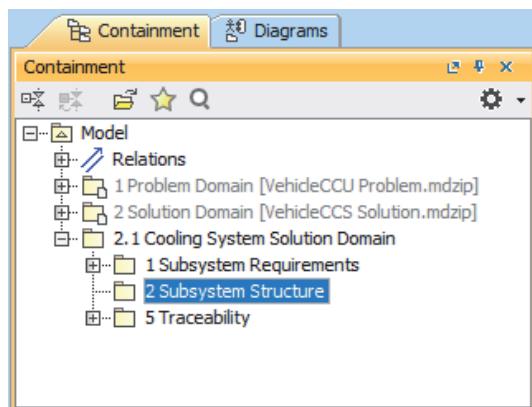


Tutorial

- Step 1. Organizing the model for subsystem structure
- Step 2. Getting ready to capture the subsystem structure
- Step 3. Capturing components of the Cooling System
- Step 4. Creating an ibd for specifying interactions
- Step 5. Specifying interactions with the outside
- Step 6. Specifying interactions within the Cooling System

Step 1. Organizing the model for subsystem structure

The LSSA of the Cooling System of the VCCS can be captured in the model you created in Chapter [Subsystem Requirements](#). To begin modeling the subsystem structure, you need to establish an appropriate structure of the model beforehand. Following the design of the MagicGrid framework, relevant model artifacts should be stored under the structure of packages, as displayed in the following figure.



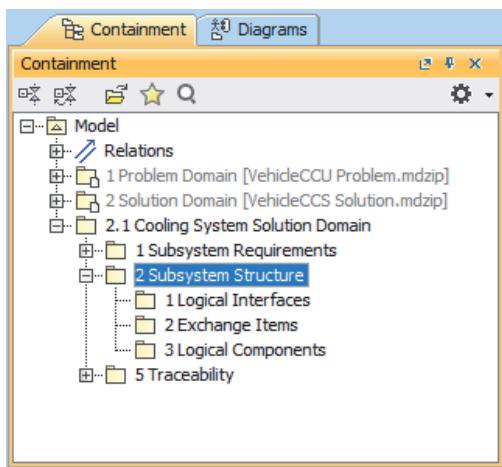
To organize the model for subsystem structure

1. Open the solution domain model of the Cooling System (the *Cooling System Solution.mdzip* file) you created in [step 1](#) of the Chapter **Subsystem Requirements** tutorial, if it is 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 Subsystem Structure* to specify the name of the new package and press Enter.

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

- 1 Logical Interfaces
- 2 Exchange Items
- 3 Logical Components

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



Step 2. Getting ready to capture the subsystem structure

Remember that the engineering team of the appointed logical subsystem must take into consideration all the interfaces of that subsystem, as determined by the system architect in the LSA model. Therefore, those interfaces must exist in the solution domain model of that logical subsystem.

Although the engineering team of the subsystem can read the LSA model, they cannot make any changes there. The first thing they should do to get ready to model the subsystem structure is create a block that captures the particular subsystem of interest (in this case, the Cooling System) and enable it to inherit all the relevant interfaces from the LSA model.

The inheritance can be achieved by establishing a generalization relationship between two blocks representing the Cooling System: the one in the LSA model (as the super-type) and the other in the LSSA model of the Cooling System (as the sub-type). As a result, the sub-type block inherits all the specifications (including proxy ports typed by appropriate interfaces) from the super-type block. It's important to note here that inherited proxy ports should be redefined; otherwise, certain tasks of building the LSSA cannot be accomplished; for example, specification of satisfy relationships from proxy ports to relevant subsystem requirements.

The generalization relationship can be created by utilizing the infrastructure of a bdd, which can be created under the *2 Subsystem Structure* package within the solution domain model of the Cooling System (created in [step 1 of this cell tutorial](#)).

To get ready to capture the structure of the Cooling System

1. Create the bdd:

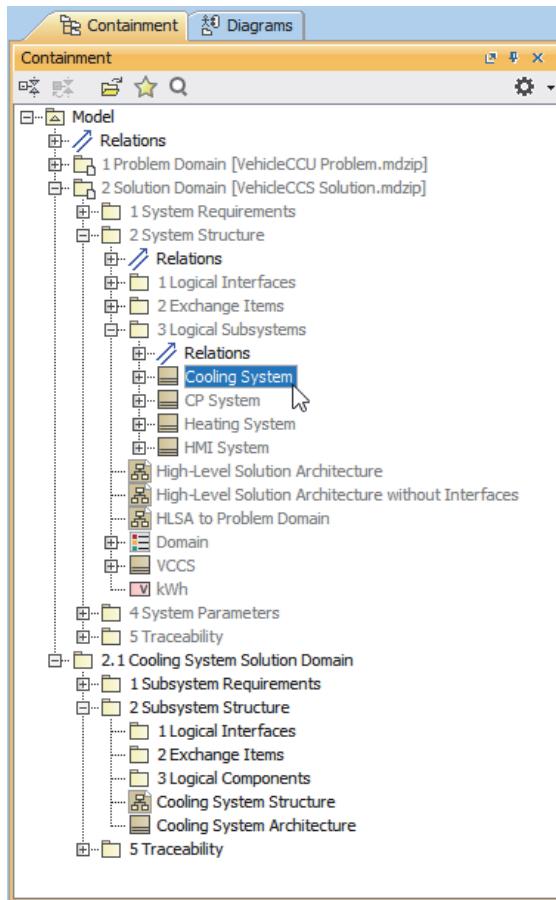
- Right-click the *2 Subsystem Structure* package and select **Create Diagram**.
- In the search box, type *bdd* (the acronym for SysML block definition diagram), and then press Enter. The diagram is created.
- Type *Cooling System Structure* to specify the name of the new diagram and press Enter again.

2. Create the block of the subsystem of interest:

- Click the **Block** button on the diagram palette and then click on the diagram pane.
- Type *Cooling System Architecture* to name the block and press Enter.

3. Display the *Cooling System* block from the LSA model on the diagram:

- Select the *Cooling System* block in the Model Browser (you may need to subsequently expand the *2 Solution Domain*, *2 System Structure*, and the *3 Logical Subsystems* packages first).

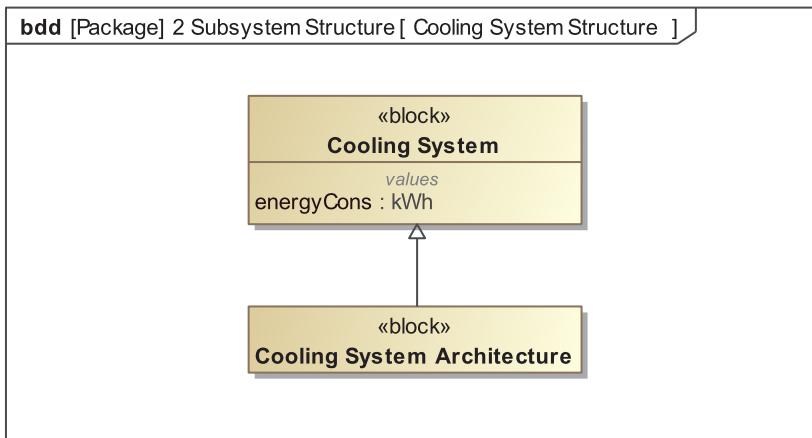


- Drag the selection to the diagram pane. The shape of the *Cooling System* block is displayed on the diagram.

4. Draw a generalization between the *Cooling System* block (super-type) and the *Cooling System Architecture* block (sub-type):

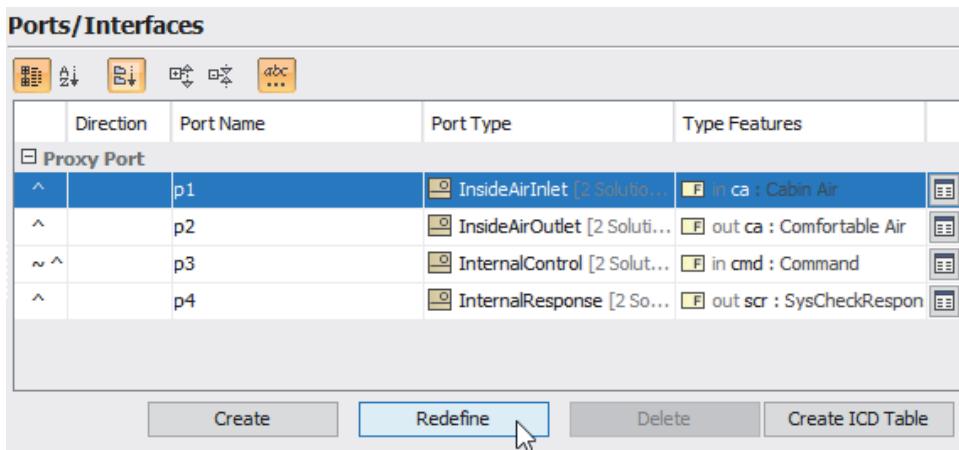
- Select the shape of the *Cooling System Architecture* block.
- Click the Generalization button on the smart manipulator toolbar of the selected block.

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

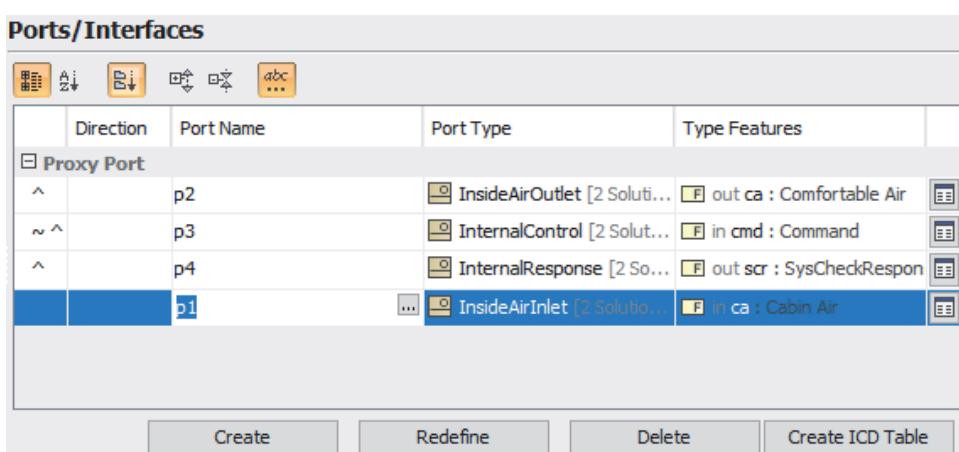


5. Redefine the inherited proxy ports:

- Double-click the shape of the *Cooling System Architecture* block on the diagram.
- On the left of the open dialog, click **Ports/Interfaces**.
- Select the proxy port in the first row and click the **Redefine** button below.



As a result, the selected proxy port is redefined and since that moment no longer decorated with the caret symbol ("^"), which indicates the inheritance. Don't change anything.



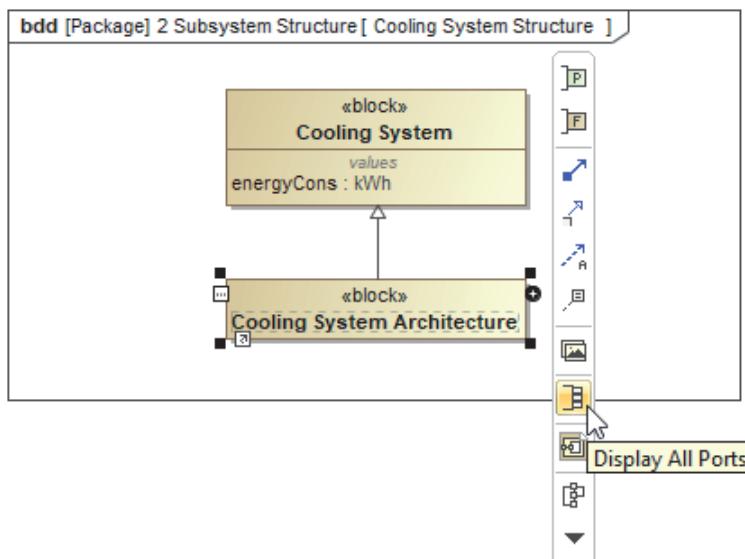
- d. Row by row, redefine the rest (the order of items may differ from what you see in the following figure).

Ports/Interfaces				
Direction	Port Name	Port Type	Type Features	
Proxy Port				
	p1	InsideAirInlet [2 Solution D...	[F] in ca : Cabin Air	
	p2	InsideAirOutlet [2 Solution...	[F] out ca : Comfortable Air	
~	p3	InternalControl [2 Solution...	[F] in cmd : Command	
	p4	InternalResponse [2 Soluti...	[F] out scr : SysCheckResponse	

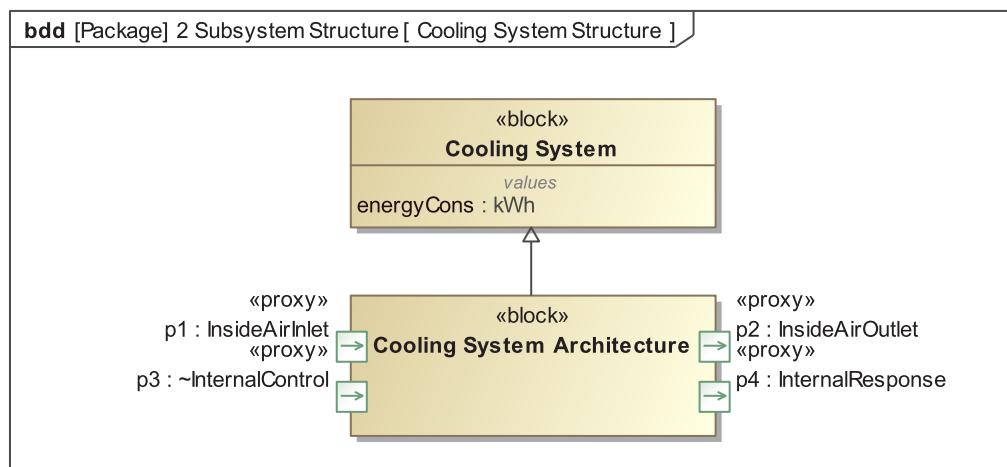
- e. Click **Close**.

6. Display the proxy ports and interfaces of the *Cooling System Architecture* block on its shape:

- Select the shape of the *Cooling System Architecture* block on the diagram pane.
- Click the Display All Ports button on its smart manipulator toolbar.



The proxy ports and interfaces are displayed.



Step 3. Capturing components of the Cooling System

As defined by the subsystem requirements, the Cooling System shall be composed of:

- Compressor
- Condenser
- Evaporator

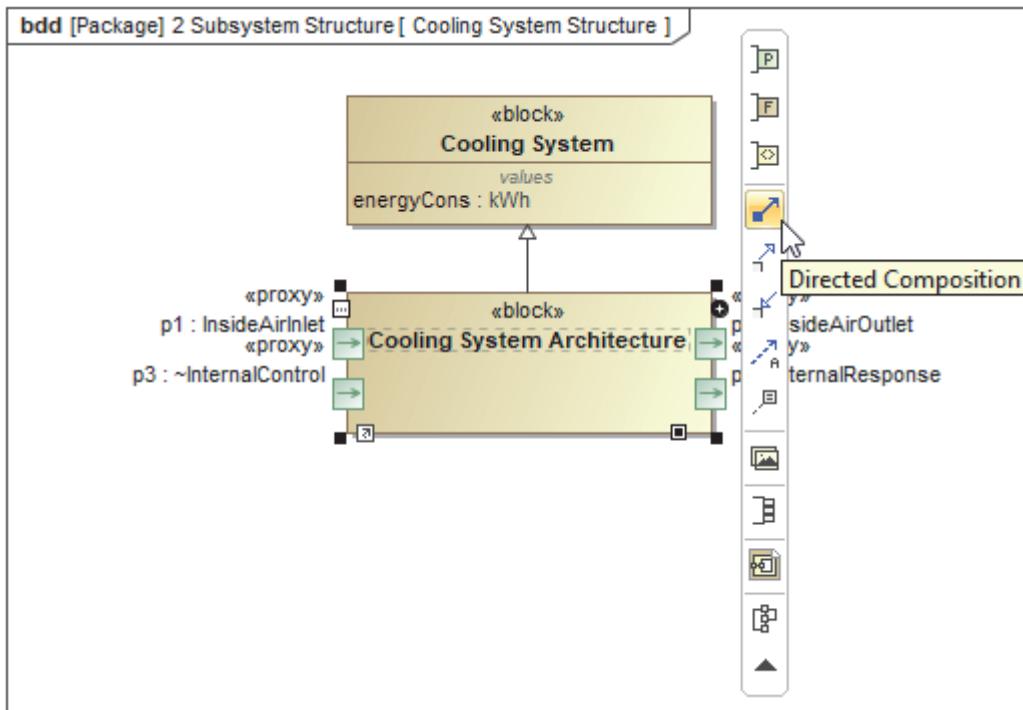
Based on their experience, the appointed engineering team identifies a few more components:

- Receiver Dryer
- Metering Device

All these items can be captured using either **bdd** or **ibd**, although we use the *Cooling System Structure* bdd (created in [step 2 of this cell tutorial](#)). The ibd is useful later, when you need to represent interactions between the specified components.

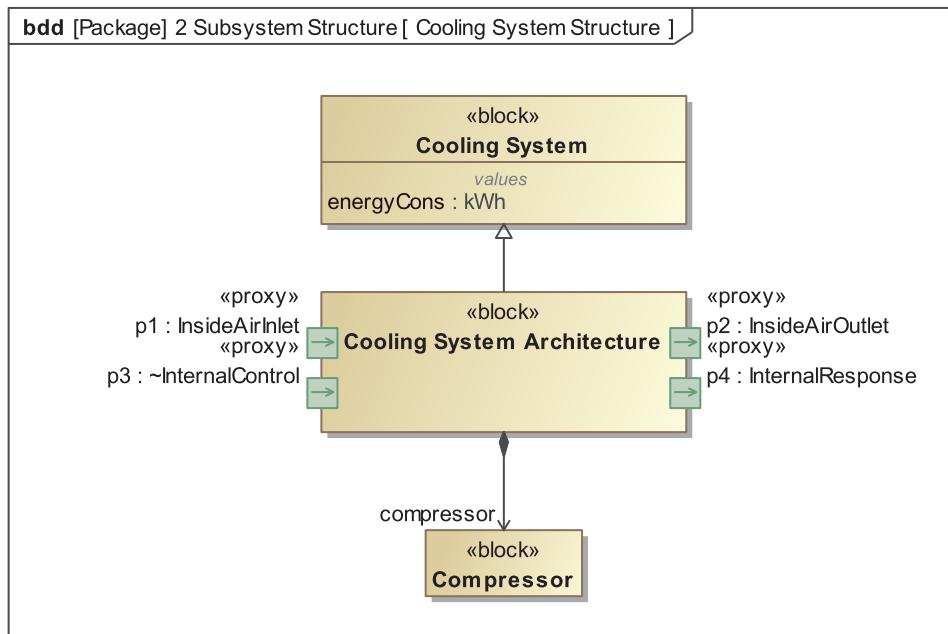
To capture the components of the Cooling System

1. Open the *Cooling System Structure* bdd, created in [step 2 of this cell tutorial](#), if not opened yet.
2. Select the *Cooling System Architecture* 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.

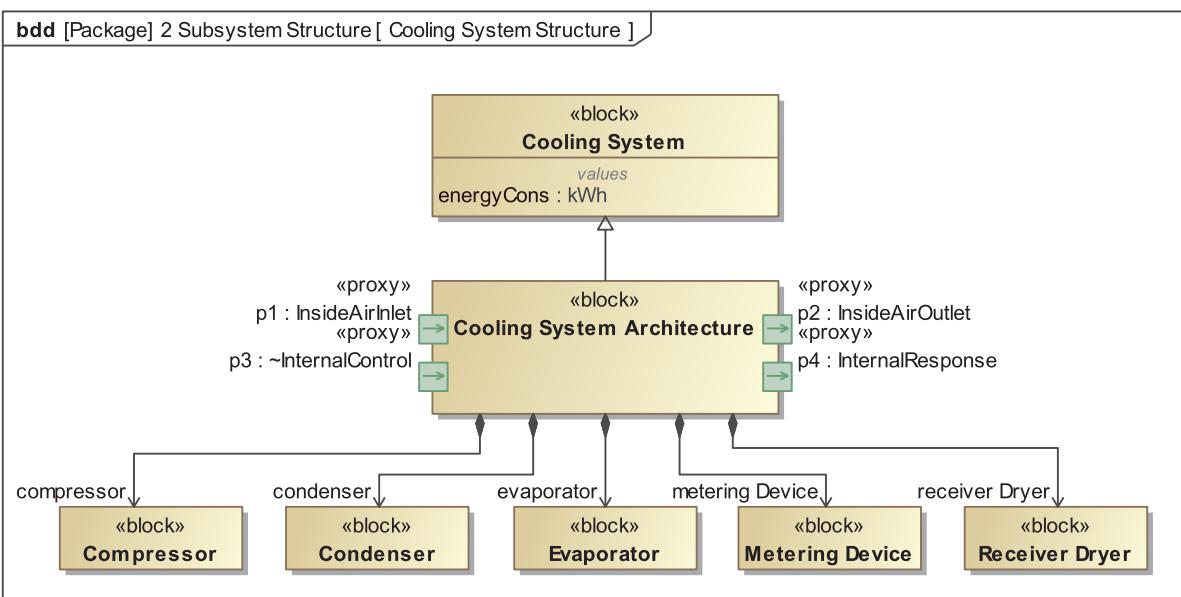


Another block is created in the Model Browser and its shape is selected on the diagram pane.

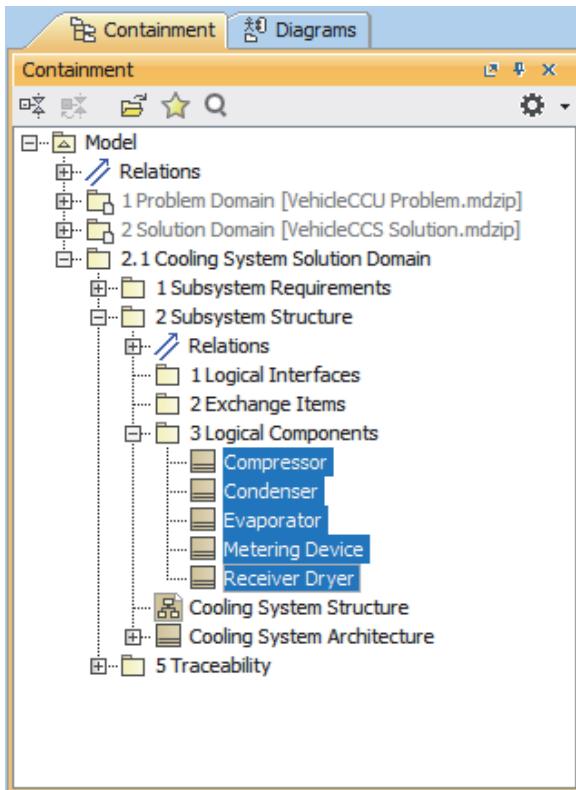
3. Type *Compressor* directly on the selected shape to specify the name of the appropriate component of the Cooling System, and press Enter.



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



5. In the Model Browser, select all the blocks of components and drag them to the *3 Logical Components* package you created in [step 1 of this cell tutorial](#). The blocks are moved to that package.

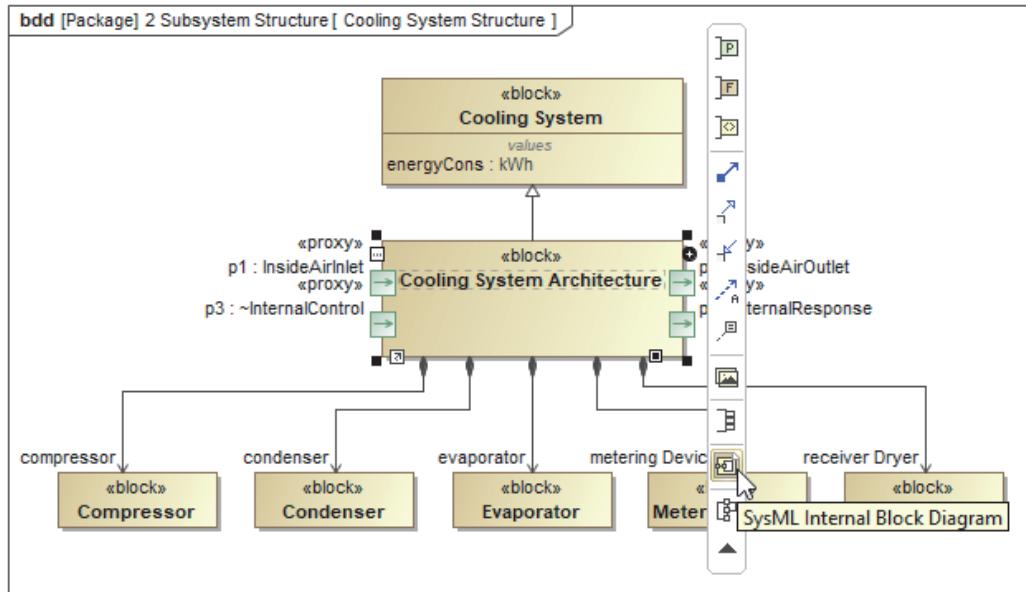


Step 4. Creating an ibd for specifying interactions

Remember that a bdd works best to capture components, while 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 Architecture* block. To begin specifying the interactions, you need to represent all proxy ports and components of the Cooling System in that diagram. They have all been captured using the *Cooling System Structure* bdd.

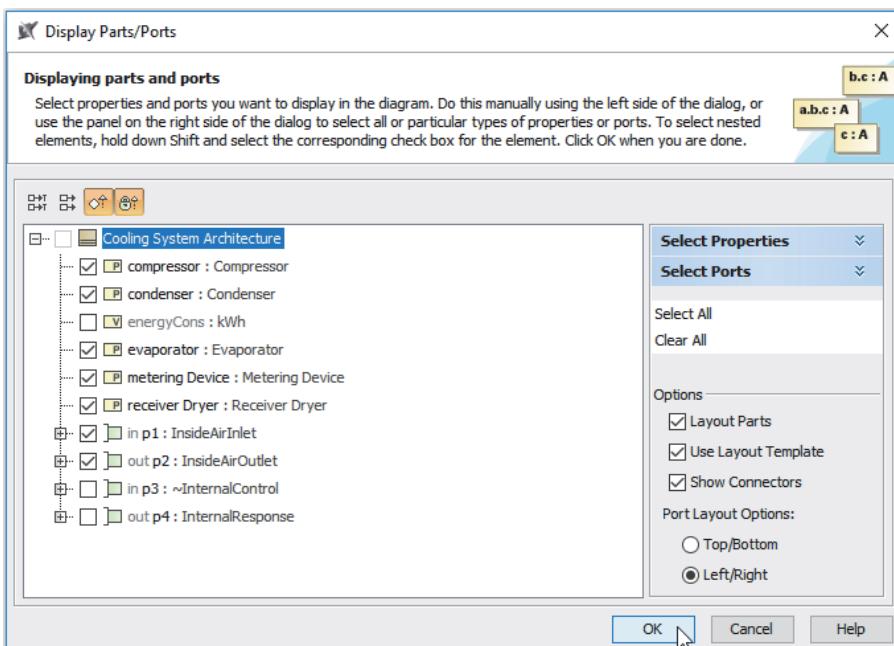
To create an ibd for specifying interactions of the Cooling System

1. In the bdd *Cooling System Structure*, select the *Cooling System Architecture* block and click the SysML Internal Block Diagram button  on its smart manipulator toolbar (see the following figure). The ibd is created and the **Display Parts/Ports** dialog opens on top of it.

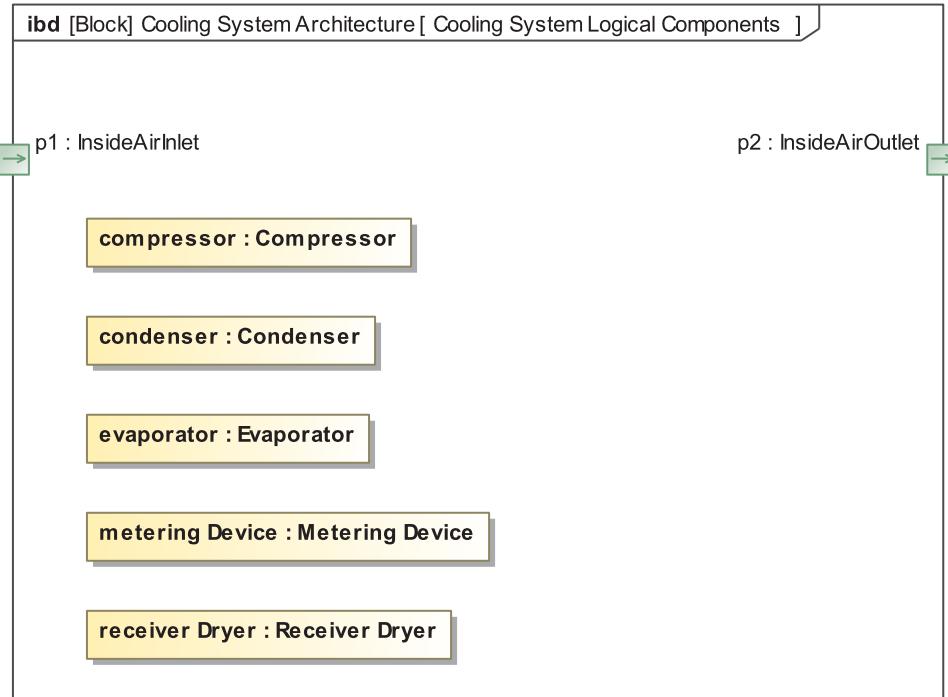


2. In the open dialog, select all the part properties and the first two proxy ports (see the following figure). Click the **OK** button.

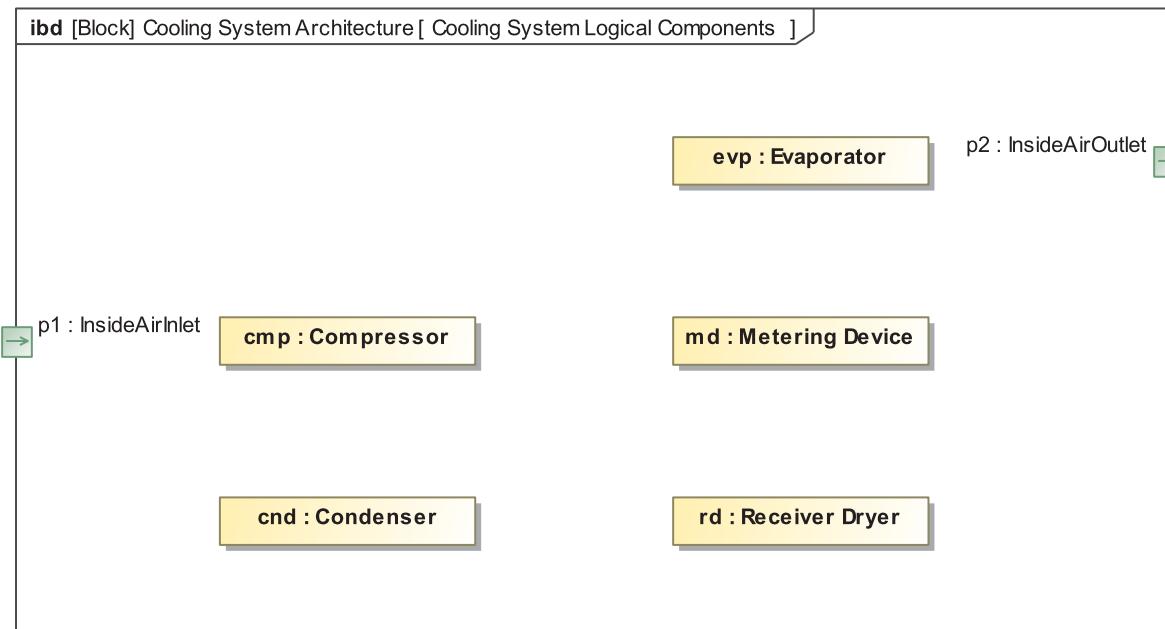
i It is not necessary to capture all communications in the same diagram. Therefore, we will skip showing a couple of proxy ports, as they are not relevant to the aspect we are going to model in the newly created diagram.



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



4. Shorten the names of part properties to make their shapes more compact (don't remove the names; you might need them later).
5. Rearrange the shapes on the diagram, as shown in the following figure.



Step 5. Specifying interactions with the outside

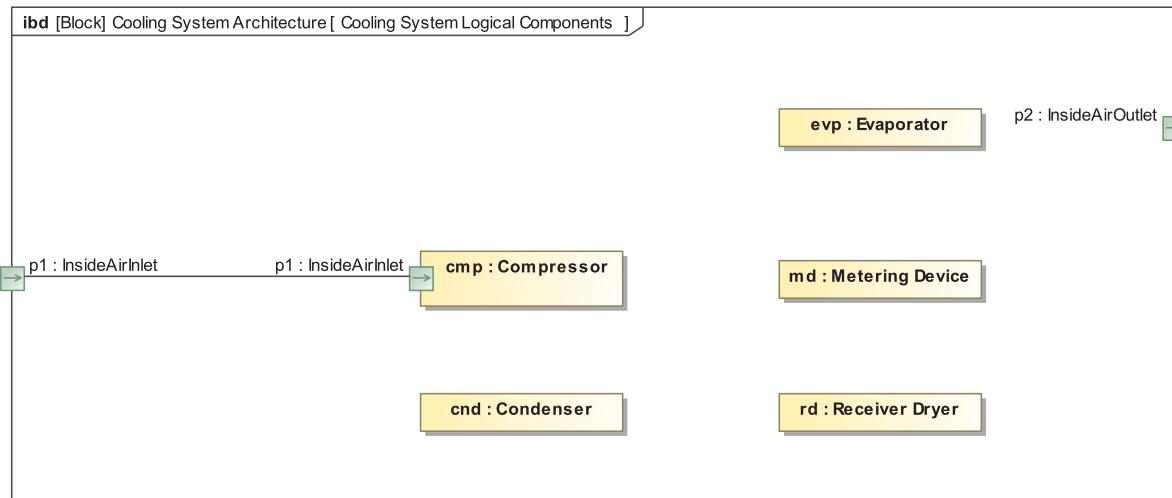
In this step, we focus on specifying interactions between components of the Cooling System and outside it; that is, other subsystems of the VCCS or even other systems of the whole vehicle. As you already know, interactions can be specified as connectors established via compatible proxy ports. Information, matter, or energy exchanged throughout these interactions can be specified as items flowing over relevant connectors.

We can start from specifying the interactions via the inherited interfaces (the ones identified in the LSA). There are two of them in this diagram: one for letting the air into the Cooling System, and the other is for

letting it out. Let's specify that the Compressor consumes the air from the cabin, and the Evaporator (its fans actually) supplies the cooled air back into the cabin.

To specify that the Compressor consumes the air from the cabin

1. Select the *p1* proxy port (typed by the *insideAirInlet* interface block) on the diagram frame and click the Connector button  on its smart manipulator toolbar.
2. Click the shape of the *cmp* part property (typed by the *Compressor* block).
3. Select **New Proxy Port**. A compatible proxy port is created for the *Compressor* block and the connector is established between the diagram frame and the selected part property.



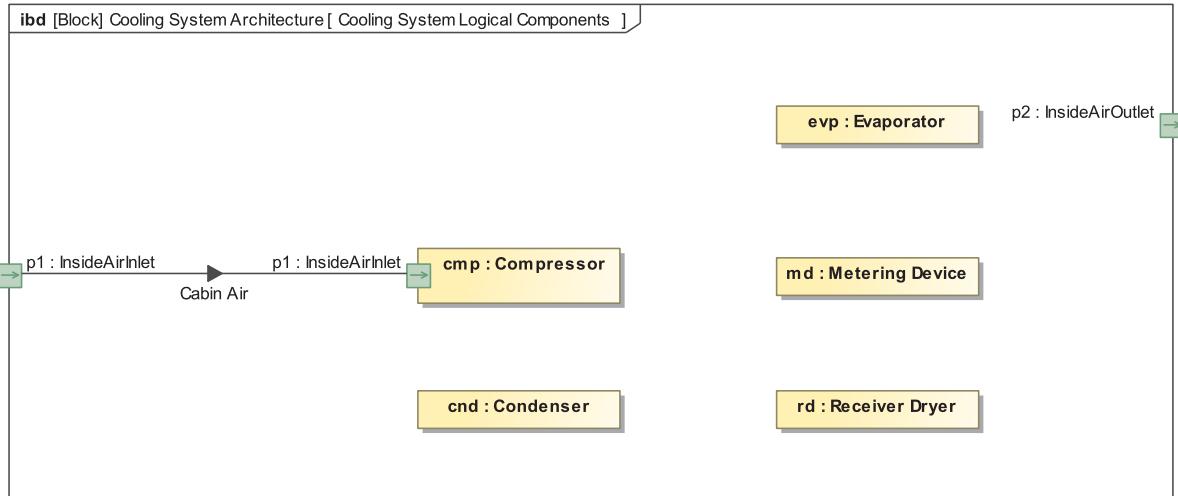
4. In the Model Browser, expand the contents of the following read-only packages: *2 Solution Domain > 2 System Structure > 2 Exchange Items*.

(i) It is important that exchange items for communication with the outside are only taken from the LSA model. Otherwise, the consistency and integrity of different logical subsystem models is broken.

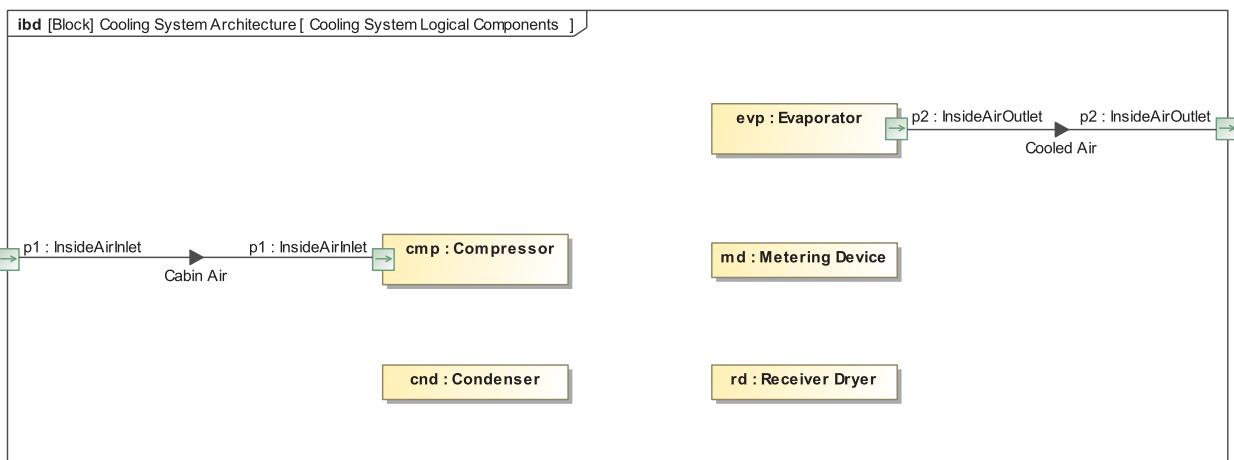
5. Select the *Cabin Air* block and drag it to the newly created connector on the open diagram pane.

(i) According to SysML, the element you want to specify as the item flowing over 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 *Cabin Air* block is specified as the item flowing over the newly created connector.



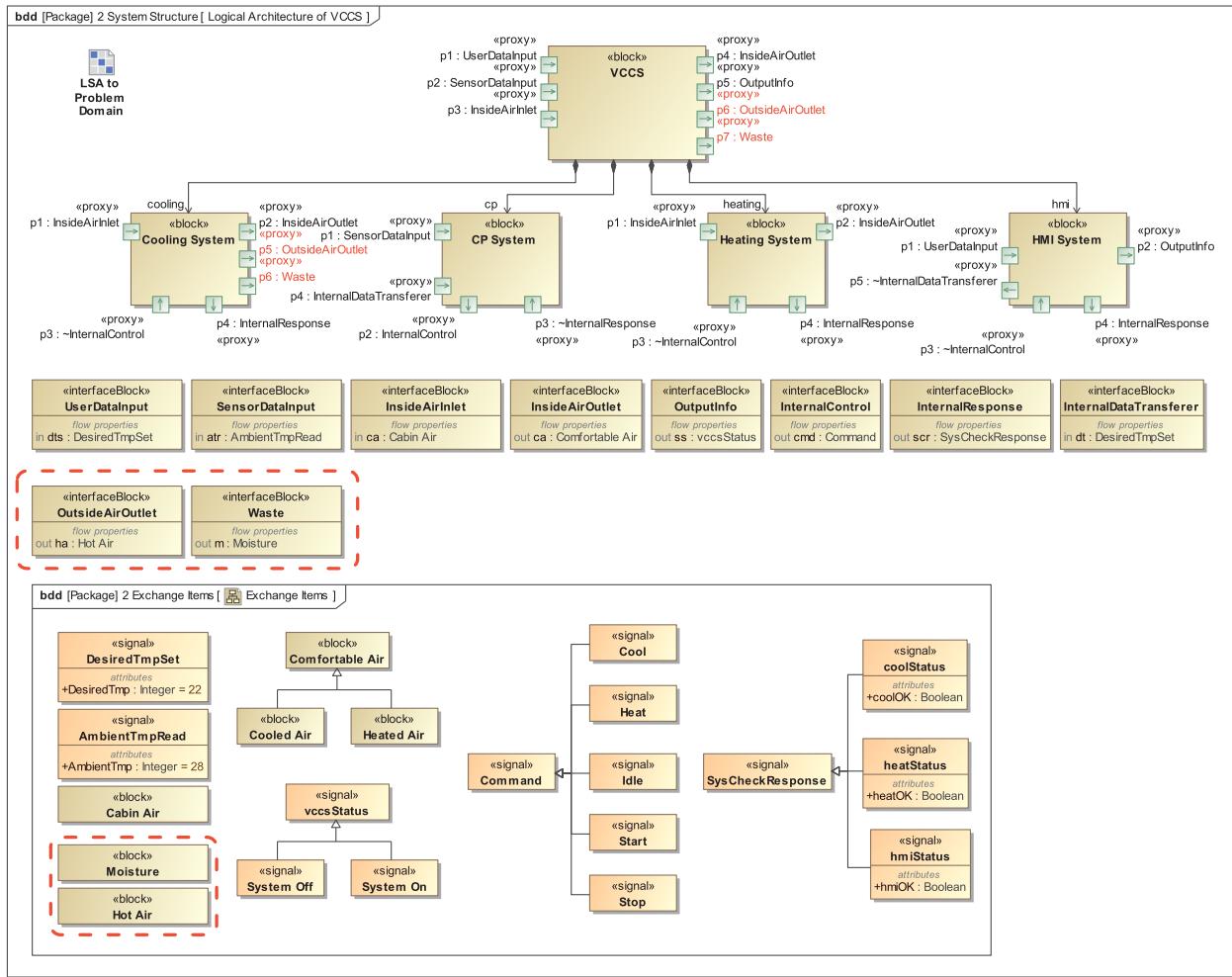
In the same way, specify that the Evaporator supplies the cooled air back to the cabin (you only need to change the flow direction in the last step). Once you're done, the ibd in your model should be similar to the diagram in the following figure.



However, we are not finished yet. Let's say that the engineering team has identified two more interfaces that are mentioned neither in the system requirements nor in the subsystem requirements specification: one for letting the hot air out the vehicle, and the other for removing the moisture separated from the refrigerant out of the VCCS. To capture this information in the model, you need to create two new interface blocks and assign them to the *VCCS* block as proxy ports. Since it is system level information, this can only be done in the LSA model. Therefore, the engineering team of the Cooling System must inform the systems architect about two new interfaces and request for relevant changes the LSA model. When this is done, the engineering team of the Cooling System can update their model to let the *Cooling System Architecture* block inherit two new proxy ports from the LSA model. These changes may also impact the work of other engineering teams; in that case, the systems architect must take the responsibility to inform them.

We assume that you temporarily get back to the LSA model and capture the mentioned interfaces on your own (this procedure is explained in Chapter [System Structure](#)). Once you're done, the *Logical Architecture of VCCS* bdd in your model should look like the diagram in the following figure (the updates are highlighted).

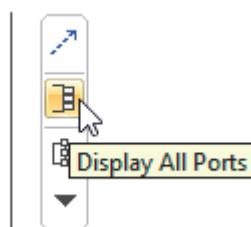
It is also important to note that the subsystem and even system requirements must be reviewed and updated accordingly. We will get back to this in Chapter [Traceability to Subsystem Requirements](#).



Now it's time to update the *Cooling System Logical Components* ibd in the solution domain model of the Cooling System. First, you need to display newly created proxy ports on the diagram frame; second, you need to specify the above mentioned interactions between the components of the Cooling System and outside it.

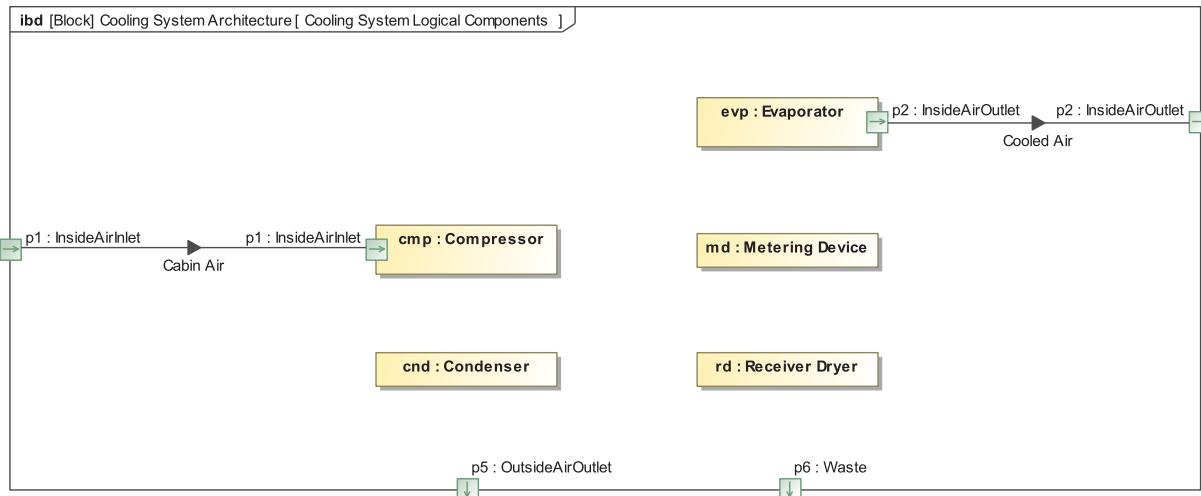
To display the newly created proxy ports on the *Cooling System Logical Components* ibd frame

1. Make sure the solution domain model (file) is updated in the solution domain model of the Cooling System.
- i** For more information on this topic, refer to the latest documentation of your **modeling tool**.
2. Open the *Cooling System Logical Components* diagram, if not opened yet.
 3. Select the diagram frame and click the Display All Ports button on its smart manipulator (see the following figure). All the proxy ports of the *Cooling System* block are displayed on the diagram frame.

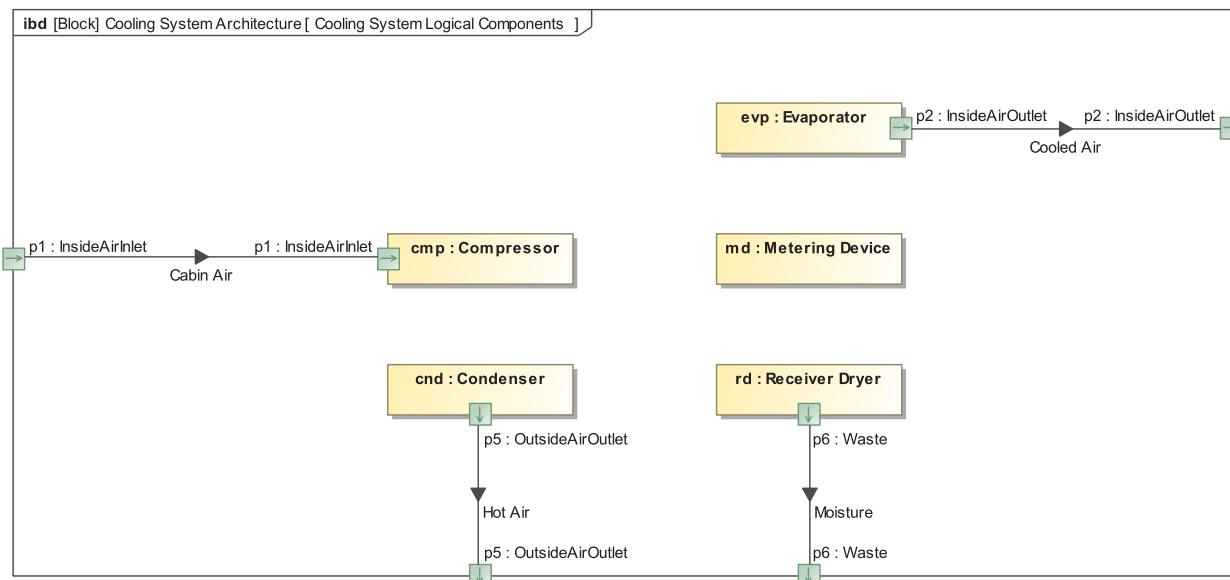


4. You don't need the ones typed by the *InternalControl* and *InternalResponse* interface blocks. Select them one by one and press Delete to remove them from the diagram (don't worry; they remain in the model).

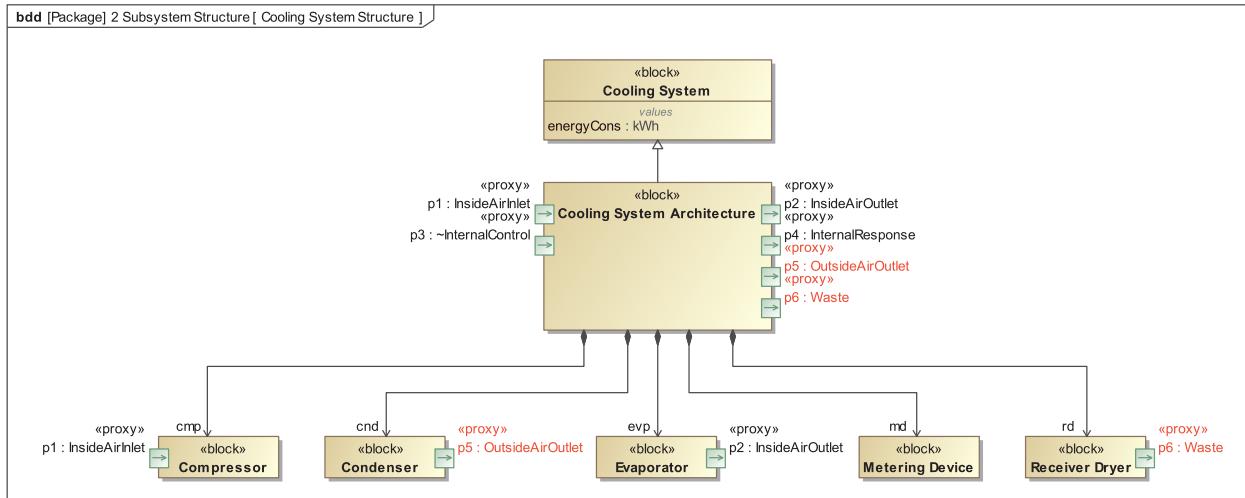
i As it was mentioned before, it is not necessary to capture all communications in the same diagram. Therefore, we will skip showing a couple of proxy ports, as they are not relevant to the aspect we are going to model in the newly created diagram.



After you capture that the Condenser releases the unneeded hot air out of the Cooling System, and the Receiver Dryer lets out the moisture, your ibd should look the same as the diagram below.



Now if you open the *Cooling System Structure* bdd created in [step 3 of this cell tutorial](#), you can update it to display new proxy ports on the blocks that capture the Cooling System and some of its components.



Step 6. Specifying interactions within the Cooling System

In this step, we proceed with specifying interactions, between different components of the Cooling System this time. As in the previous step, interactions can be specified as connectors established via compatible proxy ports. Information, matter, or energy exchanged throughout these interactions can be specified as items flowing over relevant connectors.

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

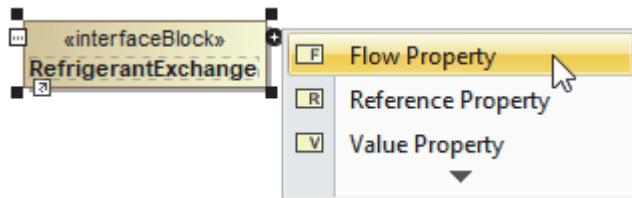
1. The Compressor consumes the refrigerant in the form of low pressure and low temperature gas and transforms it into high pressure and high temperature gas (almost liquid).
2. Inside the Condenser, the refrigerant turns into liquid.
3. The Receiver Dryer separates the moisture from the refrigerant, but doesn't change its physical characteristics.
4. The Metering Device makes the refrigerant turn into low pressure and low temperature liquid.
5. The Evaporator turns the refrigerant into gas and provides it to the Compressor to close the loop.

The list above determines one more interface of the Cooling System. It is for exchanging the refrigerant and can be captured in the model as a new interface block. The engineering team doesn't need to inform anybody about the newly identified interface; it is used for internal communication only and can be captured in the solution domain model of the Cooling System.

To capture the interface block for exchanging the refrigerant

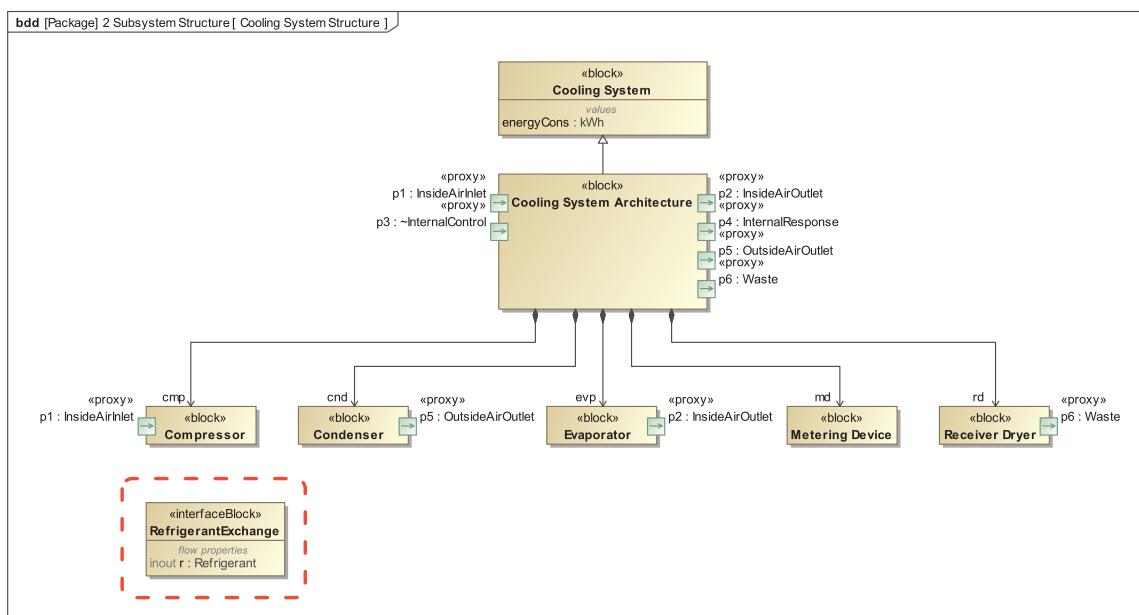
1. Open the *Cooling System Structure* diagram.
2. Click the **Interface Block** button on the diagram palette and then click an empty place on the diagram pane. An unnamed interface block is created in the model, directly under the *2 Subsystem Structure* package, and displayed on the diagram.
3. Type *RefrigerantExchange* to name the interface block and press Enter.

4. Click the Create Element button  on the interface block shape and select **Flow Property**.



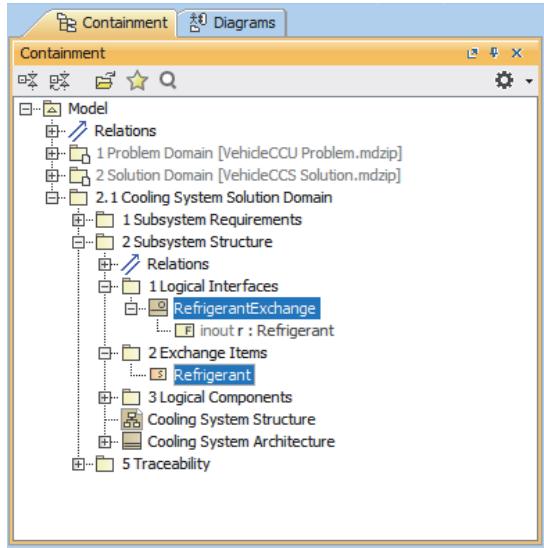
5. Directly on the shape of the interface block, do the following:

- Leave *inout*, the default direction of the flow property, as it is.
- Type *r* to specify the flow property name. Do not press Enter!
- Type *:Refrigerant* and press Enter. The *Refrigerant* signal is created in the model directly under the *RefrigerantExchange* interface block and immediately set as its only flow property type.



6. Select the *RefrigerantExchange* interface block in the Model Browser:

- Select the shape of this interface block on the diagram.
 - Press Alt + B.
7. Drag the selection to the nearest *1 Logical Interfaces* package. The *RefrigerantExchange* interface block is now stored in this package as it should be.
8. Expand the contents of the interface block and select the *Refrigerant* signal.
9. Drag the signal to the nearest *2 Exchange Items* package. The *Refrigerant* signal is now stored in this package as it should be.

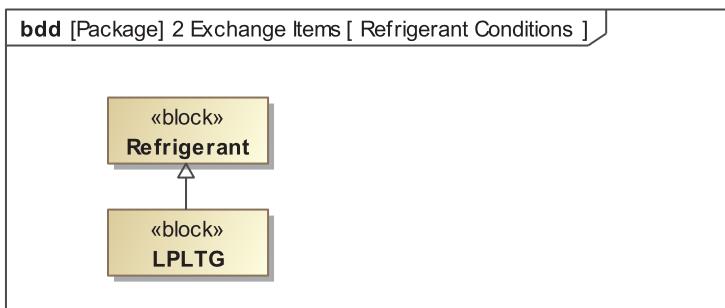


As you can see, the refrigerant was automatically captured in the model as a signal, which is not correct, as the nature of the refrigerant implies that it should be defined as a block. This can be easily fixed by refactoring the *Refrigerant* signal to a block. The next thing you need for specifying the above described interactions is diverse physical conditions of the refrigerant. Each condition can be captured as a sub-type of the *Refrigerant* block. For creating sub-types, the infrastructure of the **bdd** can be used.

To capture diverse physical conditions of the refrigerant

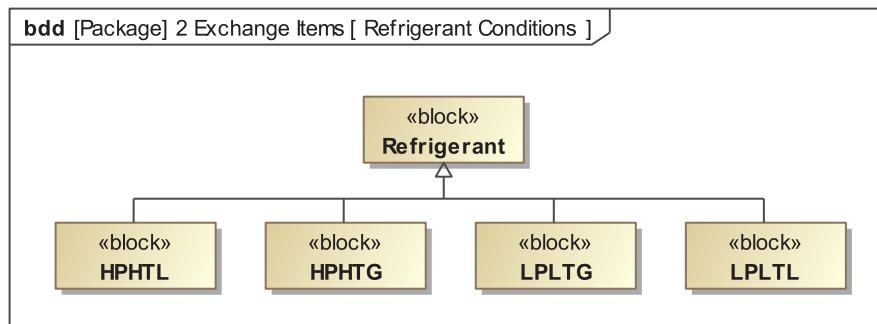
1. Create a bdd:
 - a. Right-click the *2 Exchange Items* package and select **Create Diagram**.
 - b. In the search box, type *bdd*, the acronym for SysML block definition diagram, and then press Enter. The diagram is created.
 - c. Type *Refrigerant Conditions* to specify the diagram name and press Enter again.
2. Drag the *Refrigerant* signal to the diagram pane. The shape of it is created on the diagram.
3. To refactor the *Refrigerant* signal to a block:
 - a. Right-click it and select **Refactor > Convert To > Block**.
 - b. Click **OK** to remove all properties that are incompatible between these two types.
4. Select the *Refrigerant* block and click the Generalization button on its smart manipulator toolbar.
5. Click an empty place on the diagram pane.
6. Type *LPLTG* directly on the new shape to name the block and press Enter.

LP stands for *Low Pressure*, LT for *Low Temperature*, G for *Gas*.



7. Follow steps 4 to 6 as many times as necessary to create the *HPHTG*, *HPHTL*, and *LPLTL* blocks.

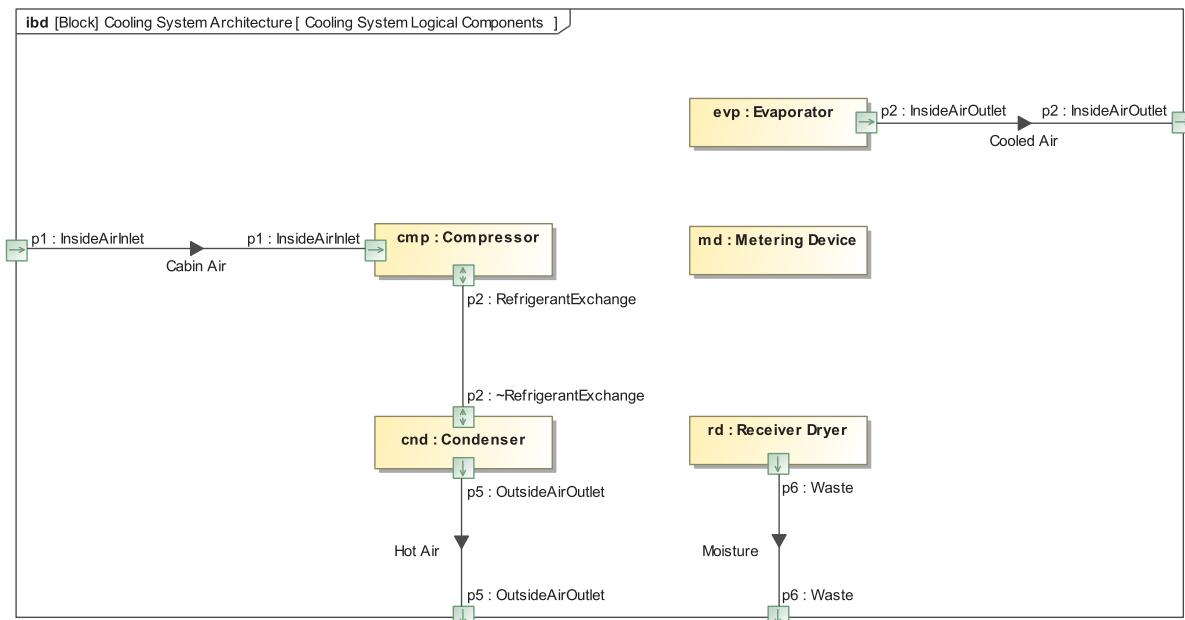
(i) HP stands for *High Pressure*, HT for *High Temperature*, L for *Liquid*.



When you're done, you can specify the interactions between the components of the Cooling System (listed at the beginning of this step).

To specify that the Compressor provides the Refrigerant to the Condenser, in the form of high pressure and high temperature gas

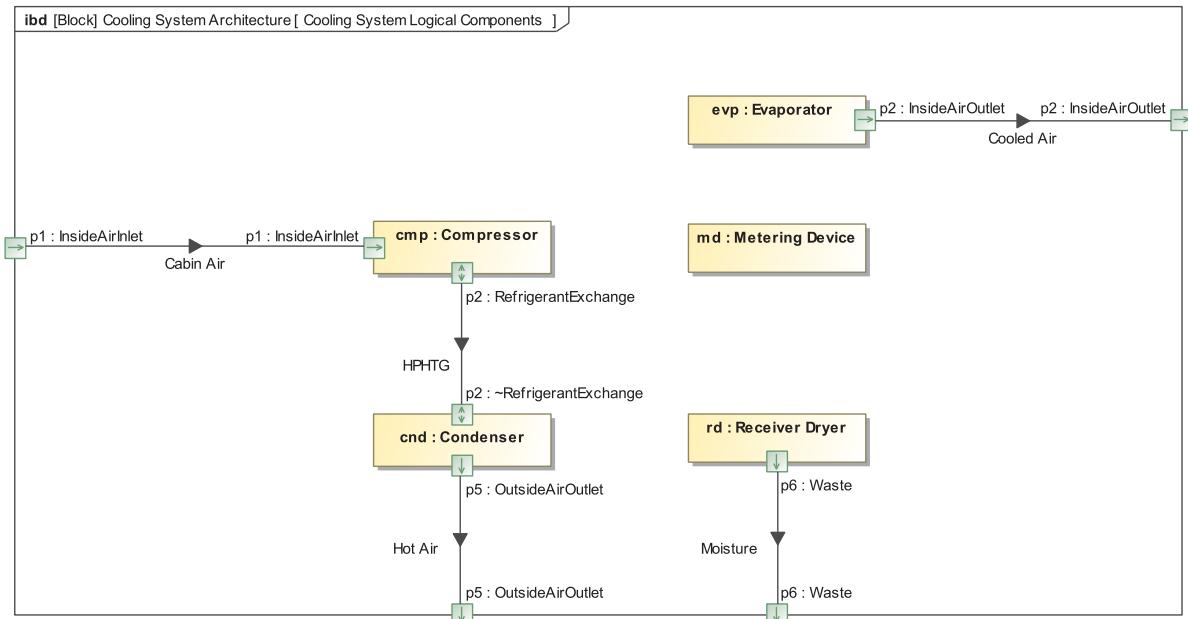
1. Make sure the Type Selection Mode is enabled in the *Cooling System Logical Components* diagram. Otherwise, creation of a proxy port doesn't trigger selection of the interface block to type that proxy port.
2. Select the shape of the part property typed by the *Compressor* block and click the Proxy Port button on its smart manipulator toolbar. A new proxy port appears on the border of the part property shape.
3. Type *ref* to find the *RefrigerantExchange* 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 *Condenser* 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 *HPHTG* block, and drag it to the newly created connector.

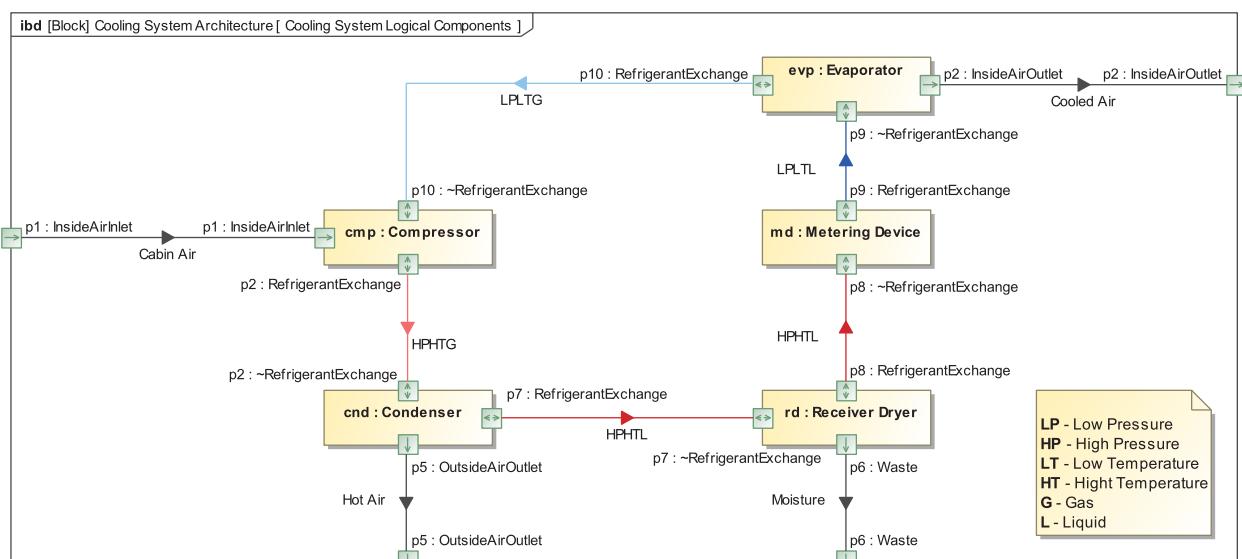
i According to SysML, the element you want to specify as the item flowing via the connector must be the type of at least one of the flow properties owned by the interface block that types the connected proxy ports. This also works with sub-types. That's why the *Refrigerant* block as well as any of its types (in this case, the *HPHTG* block), can be assigned to the connector.

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

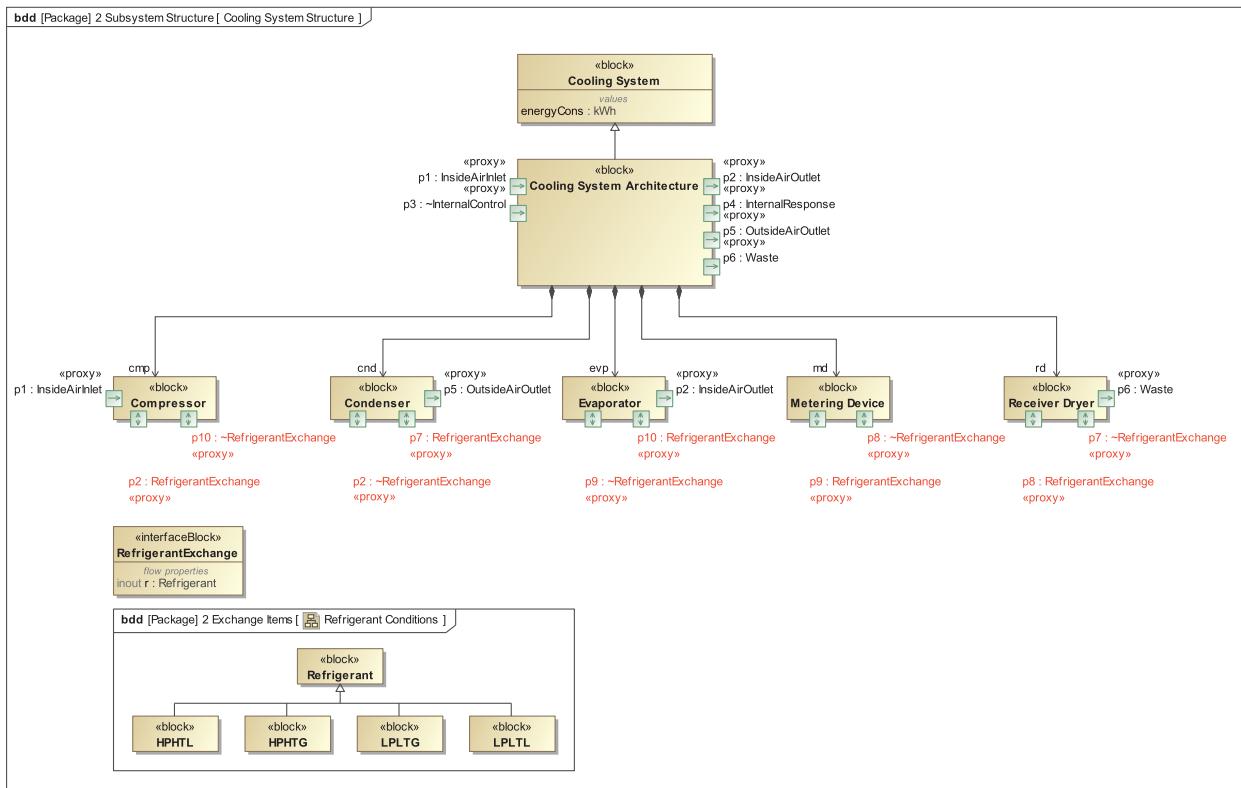


In the same way, specify the rest of the previously described interactions. Once you're done, your ibd should be similar to the one in the following figure.

- i** The connector colors can be changed by modifying their symbol properties. 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.



Now if you get back to the *Cooling System Structure* bdd created in **step 3 of this cell tutorial**, you can update it with the newly created proxy ports.



Subsystem Structure done. What's next?

- Having logical components and knowing how they communicate, you can proceed with structure modeling and go inside each logical component.
- However, there are still two unvisited cells of the solution architecture for the subsystem. That's why you should proceed to Chapters **Subsystem Behavior** and **Subsystem Parameters**.

Subsystem Behavior

What is it?

Once finished with the subsystem structure, the appointed engineering team can switch to modeling the behavior of that subsystem. Behavior models can be relatively abstract and must satisfy functional requirements of that subsystem (see Chapter [Subsystem Requirements](#)). Using the capabilities of the [modeling tool](#), these models can be executed against test cases to show whether or not these functional requirements are satisfied.

When finished, the subsystem behavior should be integrated with other subsystem behaviors, into a single model that represents the comprehensive behavior of the whole [Sol](#). This can be achieved only if the integrated models are compatible; that is, if they can communicate over interfaces and exchange items. Remember that interfaces and exchange items are determined by the systems architect in the LSA model.

This cell can be skipped if the engineering team makes the decision to model the behavior of each logical component captured in the structure model of that logical subsystem (see Chapter [Subsystem Structure](#)).

Note that the following material mainly focuses on building the behavioral model of the Cooling System of the VCCS. In the final steps of this cell tutorial, the scope extends to include the behavioral model of the CP System. This is necessary for learning how to model the communication between the two subsystems. Behavioral models of other subsystems, like Heating System and HMI System, are not included; however, you should imagine that appointed engineers / engineering teams work on them in parallel.

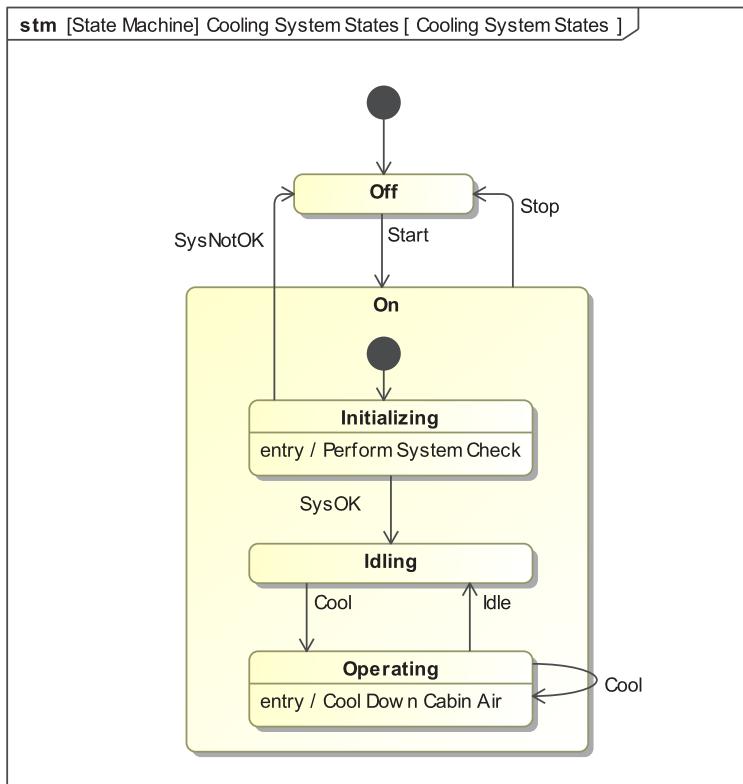
Who is responsible?

The behavior of the appointed logical subsystem can be captured by the Systems Architect, or the Systems Engineer who belongs to the engineering team responsible for building the solution architecture of that logical subsystem.

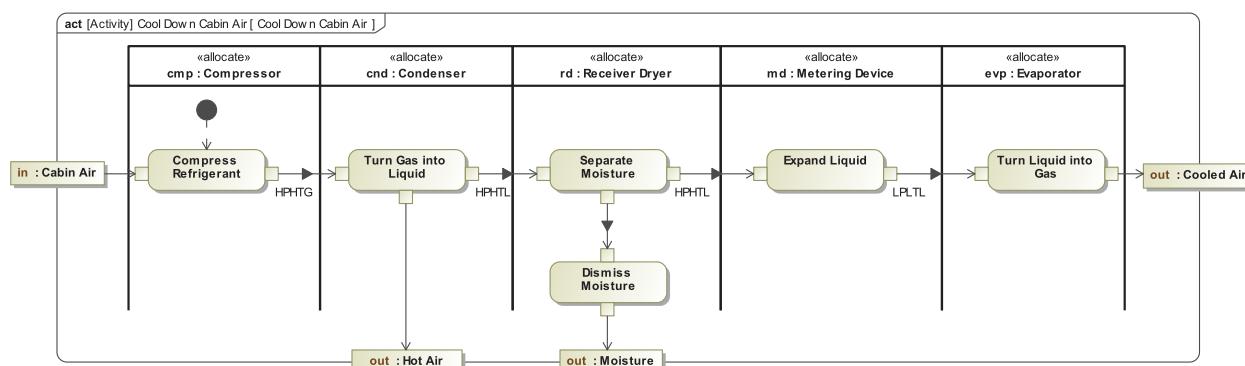
How to model?

The behavior model of the subsystem can be captured by utilizing the infrastructure of the SysML state machine and activity or sequence diagrams in combination. The state machine allows for capturing states of the subsystem and transitions between them in response to event occurrences over time. The activity or sequence diagrams 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 (i.e., a signal event such as *Start* between the *Off* and *On* states). These signals are taken from the LSA model. This ensures the integrity of behaviors models of diverse logical subsystems.



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 *Operating* state has the *Cool Down Cabin Air* activity specified as the *entry* behavior. The *Cool Down Cabin Air* activity diagram is displayed as follows.



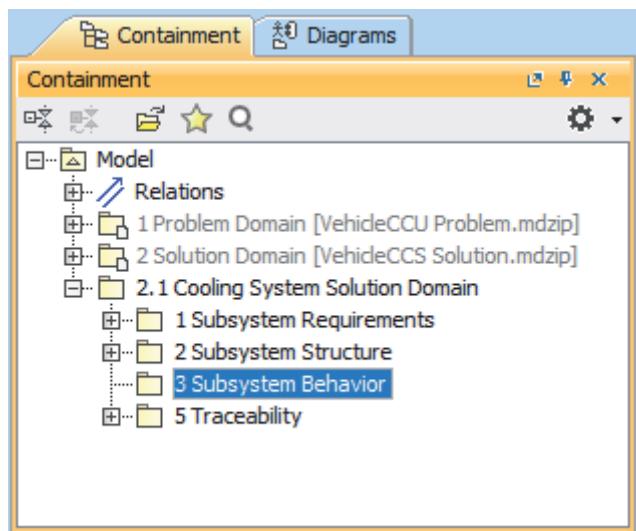
Tutorial

- Step 1. Organizing the model for subsystem behavior
- Step 2. Creating a diagram for capturing Cooling System states
- Step 3. Capturing states of the Cooling System
- Step 4. Specifying event occurrences on transitions
- Step 5. Specifying internal behaviors of the state
- Step 6. Synchronizing item flows from the structure model

Step 1. Organizing the model for subsystem behavior

The states of the Cooling System can be captured in the solution domain model of that subsystem (the one you created in Chapter [Subsystem Requirements](#)). According to SysML, states can only be stored under the block whose behavior they represent. In this case, it is the *Cooling System Architecture* block; thus, you don't need to create any additional packages for storing states in the model. They should appear directly under the *Cooling System Architecture* block, which is stored within the *2 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 the subsystem behavior

1. Open the solution domain model of the Cooling System (the *Cooling System Solution.mdzip* file) you created in [step 1](#) of the Chapter [Subsystem Requirements](#) 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 *3 Subsystem Behavior* to specify the name of the new package and press Enter.

Step 2. Creating a diagram for capturing Cooling System states

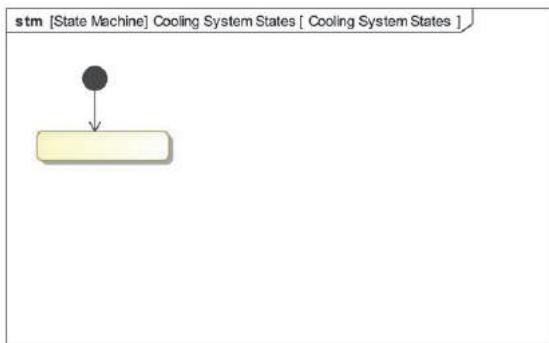
The SysML state machine diagram that describes the states of the Cooling System should be created directly under the block which captures that subsystem.

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 Chapter [Subsystem Requirements](#) tutorial, if not opened yet.

2. In the Model Browser, select the *Cooling System Architecture* block. For this, use the quick find capability:
 - a. Press Ctrl + Alt + F. The **Quick Find** dialog opens.
 - b. Type *csd*, *c* for *Cooling*, *s* for *System*, and *a* for *Architecture*.
 - c. When you see the *Cooling System Architecture* block selected in the search results list below, press Enter. The *Cooling System Architecture* block is selected in the Model Browser.

3. Create the SysML state machine for the *Cooling System Architecture* block:
 - a. Right-click the *Cooling System Architecture* 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:

1. Off
2. On:
 - a. Initializing
 - b. Idling
 - c. Operating

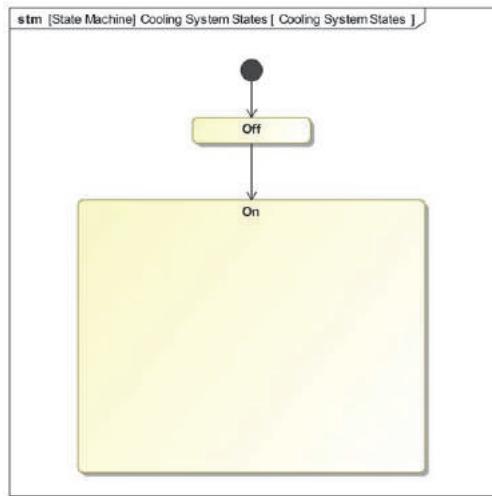
The following rules describe transitions between those states:

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

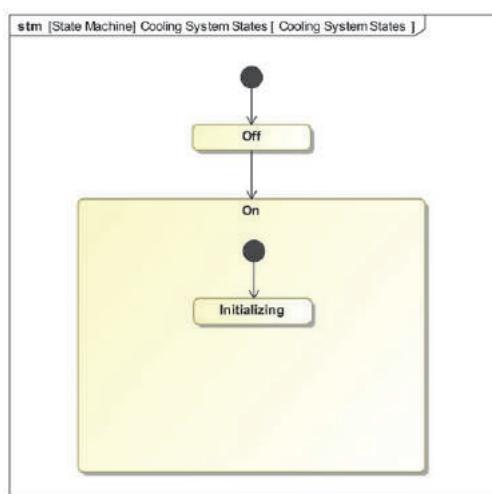
Event occurrences that trigger transitions between states are discussed later in this tutorial. This step mainly focuses on creating states and transitions between them.

To specify states of the Cooling System

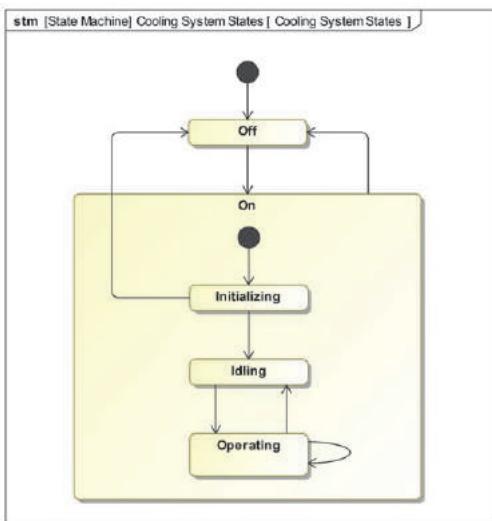
1. Open the *Cooling System States* diagram, if not opened yet. The diagram already contains the initial state and an unnamed state.
2. Select the shape of the unnamed state and then click somewhere in the middle of that shape. The shape switches to the name edit mode.
3. Type *Off* to specify the name 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

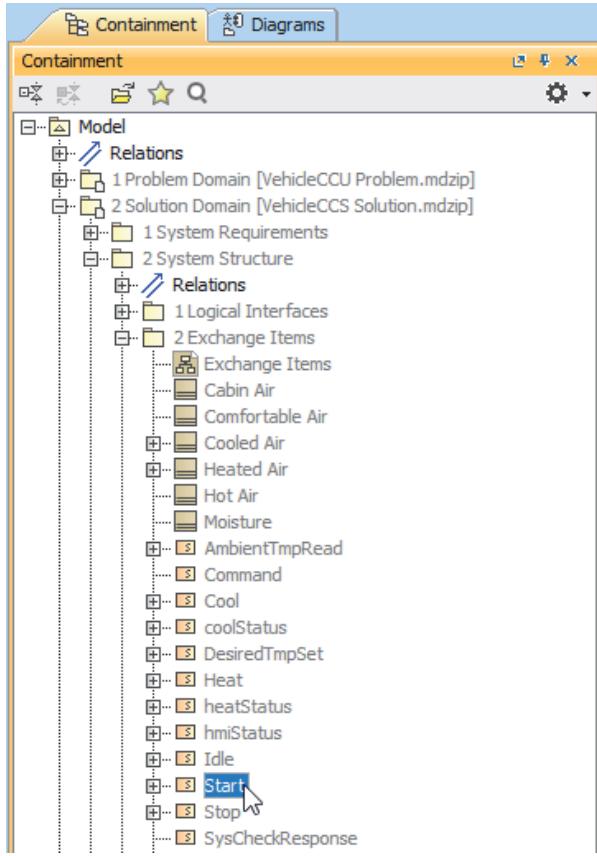
As defined by SysML, transitions between states can be triggered by various types of events, such as time, change, signal events, etc. Transitions between states of the Cooling System are triggered by signal events only. For example, the Cooling System moves from the *Off* to the *On* state after it receives the *Start* signal; it moves back to the *Off* state when the *Stop* signal arrives.

It's important to note that these signals come from outside the Cooling System. They actually come from the CP System, to be more precise. Therefore, the engineering team doesn't need to capture them in the LSSA model of the Cooling System. These signals have already been captured in the LSA model and therefore must be taken from there. The engineering team that works on the LSSA of the CP System uses these signals in their model, too. The only difference is that the CP System sends them to the outside.

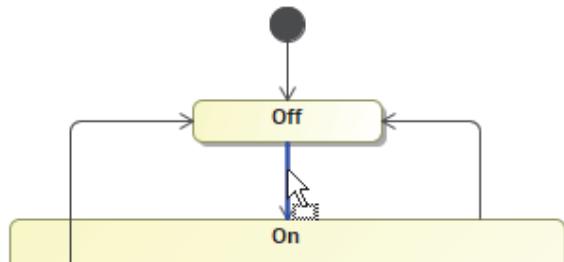
Let's start from specifying the trigger on the transition between the *Off* and *On* states.

To make the *Start* signal a trigger of the transition between the *Off* and *On* states of the Cooling System

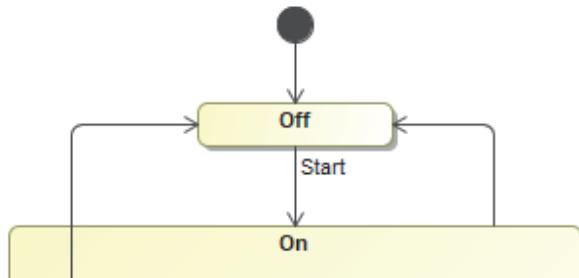
-
1. Open the *Cooling System States* diagram, if not opened yet.
 2. In the Model Browser, expand the contents of the following read-only packages: *2 Solution Domain > 2 System Structure > 2 Exchange Items*.



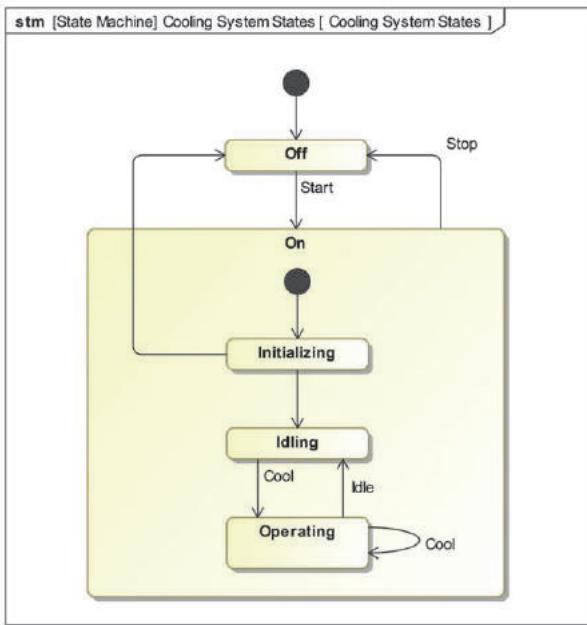
3. Select the *Start* signal and drag it to the transition between the *Off* and *On* states.



4. When you release the mouse, the *Start* signal becomes the signal event that triggers the transition between the *Off* and *On* states of the Cooling System.



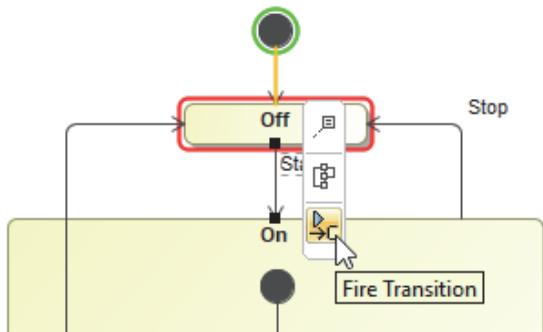
Looking at the following figure, specify *Stop*, *Cool*, and *Idle* triggers.



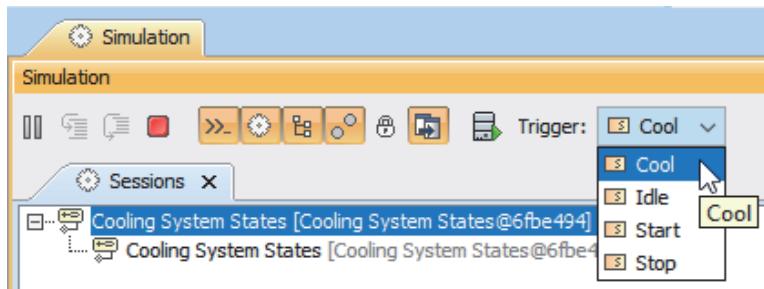
When you're done, you can run the simulation to validate the specified behavior and see if your model is correct. To start the simulation session, click the Run button on the *Cooling System States* diagram toolbar. In the **Simulation** panel, click the Start button to start the machine execution. Note that the state machine enters the *Off* state and rests there waiting for the *Start* signal to trigger it.

There are two places from where you can fire the transition:

- On the diagram



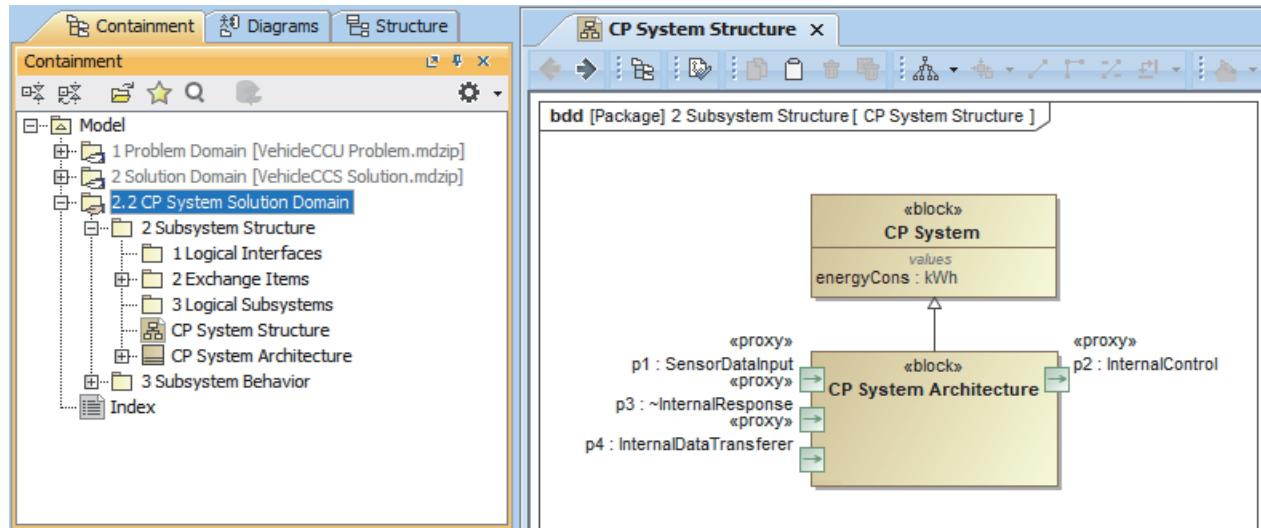
- On the **Simulation** panel



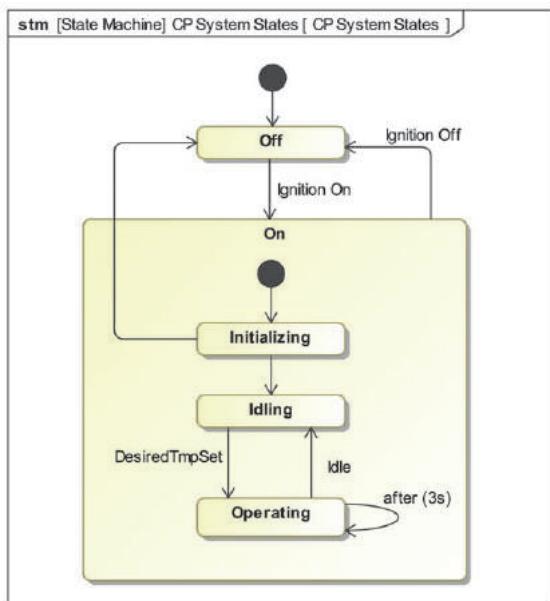
Try to fire all the transitions and see if all states can be visited. As you may see, a couple of transitions still have no triggers. These will be specified later, following a certain logic of subsystems communication.

Now it's time to imagine yourself being the engineering team who is responsible for the LSSA of the CP System, and create the state machine capturing the behavior of that logical subsystem. This is necessary

to be able to model the communication between the two subsystems. Just like the LSSA of the Cooling System, the one of the CP System should be kept in a separate model. We assume that you create the model, use in it the LSA model, and establish the necessary package structure on your own (see [step 1](#) of the [Subsystem Requirements](#) tutorial, to refresh your memory). Afterwards, create a bdd for the logical subsystem architecture and a block to capture the CP System. Then make it a sub-type of the block that represents the CP System in LSA model (see [step 2](#) of the [Subsystem Structure](#) tutorial) and finally redefine inherited from it proxy ports.



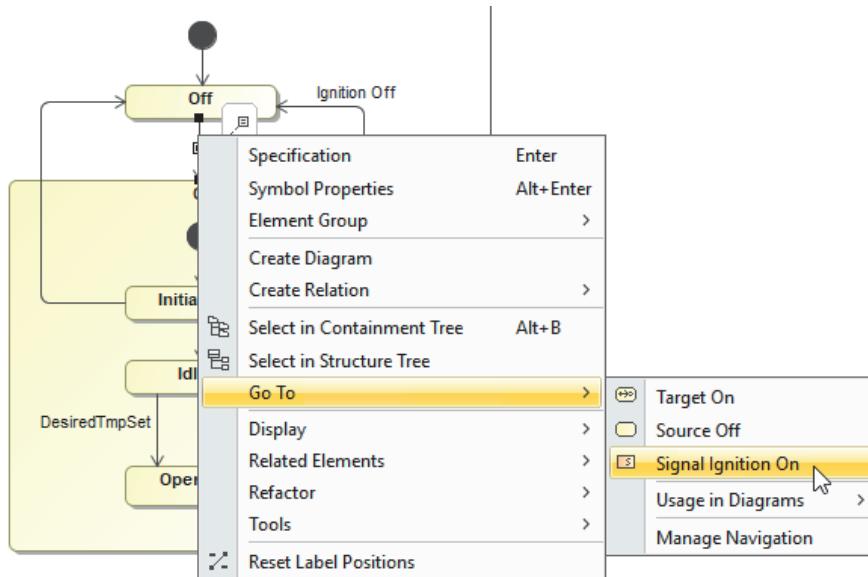
When you have the model you see in the preceding figure, you can specify the behavior of the CP System. Note that the *Ignition On* and *Ignition Off* signals are captured in the LSSA model of the CP System, while the *DesiredTmpSet* and *Idle* signals are taken from the LSA model.



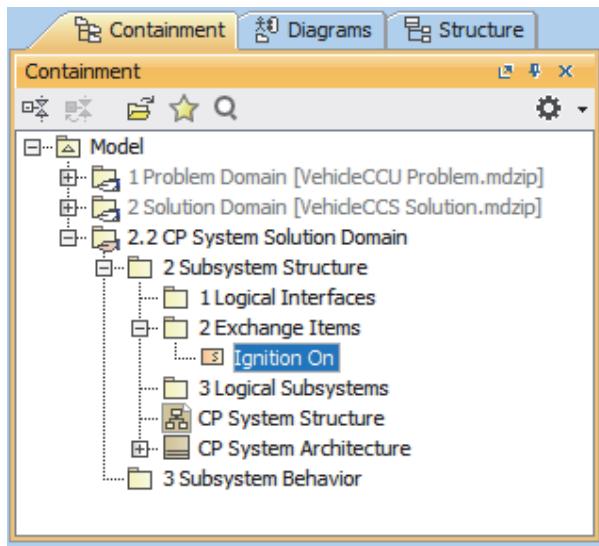
To make the *Ignition On* signal a trigger of the transition between the *Off* and *On* states of the CP System

1. Select the transition from the *Off* to *On* state.
2. Type *Ignition On* directly on the path and press Enter. The *Ignition 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. Right-click the transition and select **Go To > Signal Ignition On** (see the following figure). The signal is selected in the Model Browser.



4. Drag the signal to the nearest *2 Exchange Items* package.



Follow the steps above to create the *Ignition Off* signal.

As you can see, the *Operating* state transition to self is triggered by another type of event: a relative time event. It allows the modeler to specify that every 3 seconds the CP System moves back to the *Operating* state. There will be an internal behavior assigned to this state, performed every time it enters the *Operating* state.

To make the time event a trigger of the transition from the *Operating* state to itself

1. Select the transition.
2. Type *after (3s)* directly on the path and press Enter. The relative time event is created on the transition to self.

3. Make sure the transition to self is still selected and press Enter. See the trigger information in the Specification of the transition.

Trigger	
Event Type	TimeEvent
Trigger	<input type="checkbox"/> Trigger:after (3s) [2.2 CP System Solution Domain::2 Subsystem St... <input checked="" type="checkbox"/> TimeEvent after (3s) [2.2 CP System Solution Domain::2 Subsystem...
Event Element	
Name	
When	3s
Is Relative	<input checked="" type="checkbox"/> true
Element ID	_19_0_3_3b4012e_1586169578311_435113_46078
Documentation	
Port	

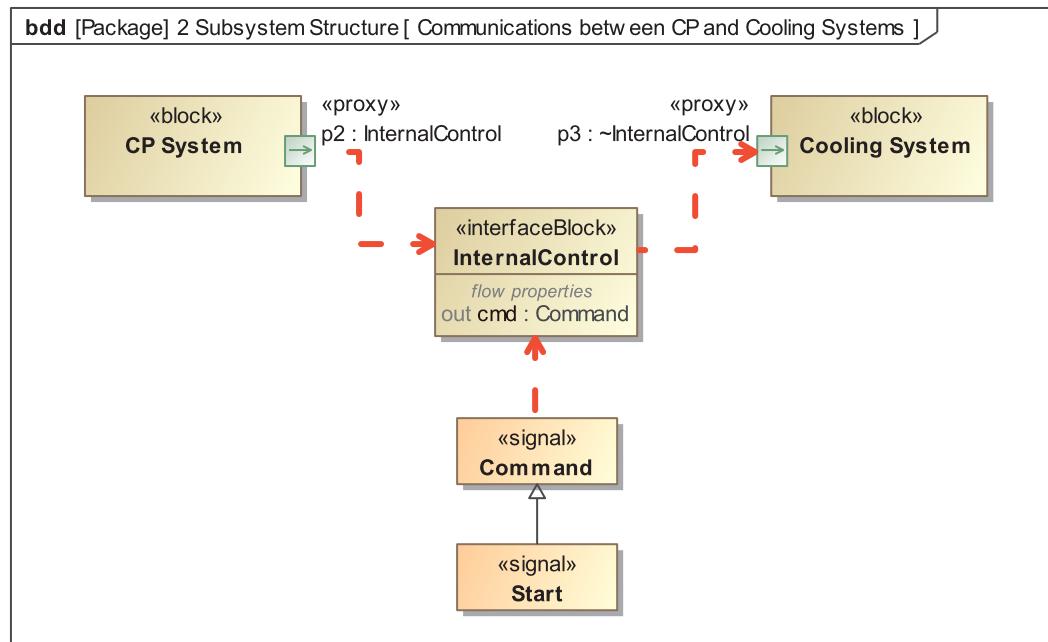
After you're done with states of the CP System, run the simulation and see if the behavior model is correct.

You have now created two state machines: one for the Cooling System and the other for the CP System. Moreover, you have specified that the Cooling System is waiting for signals from the CP System. Now it's time to capture when and how the CP System sends them.

Step 5. Specifying internal behaviors of the state

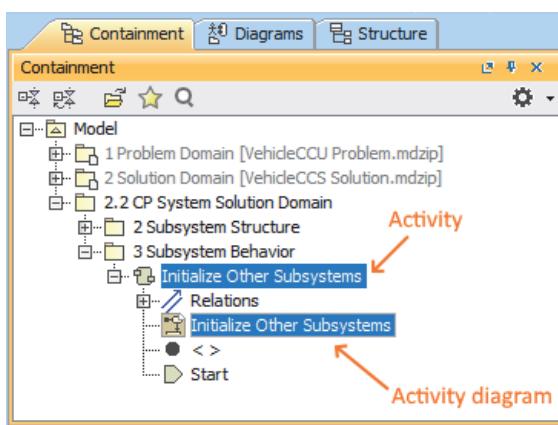
Every logical subsystem of the **Sol** can perform one or more behaviors related to one or more of its states. These behaviors are generally referred as internal behaviors of the state. The internal behavior can be captured in the form of the SysML activity diagram and stored in the relevant LSSA model, under the **Subsystem Behavior** package (see [step 1 of this tutorial](#)).

Let's say the appointed engineering team has to specify that when the CP System enters the *Initializing* state, it invokes other logical subsystems of the VCCS. Therefore, it should send the *Start* signal to all of them. Since the signal should be sent to outside the CP System, it is important to capture that it is passed via the *p2* proxy port, as shown in the following figure.

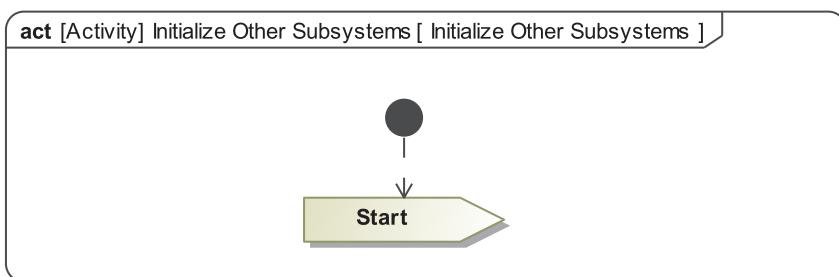


To specify that the CP System sends out the *Start* signal on entering the *Initializing* state

1. Open the LSSA model of the CP System (the one you created in the previous step), if not opened yet.
2. Create a new SysML activity diagram:
 - a. In the Model Browser, select the *3 Subsystem Behavior* package.
 - b. Right-click the package and select **Create Diagram**.
 - c. 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.
 - d. Type *Initializing Other Subsystems* 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.

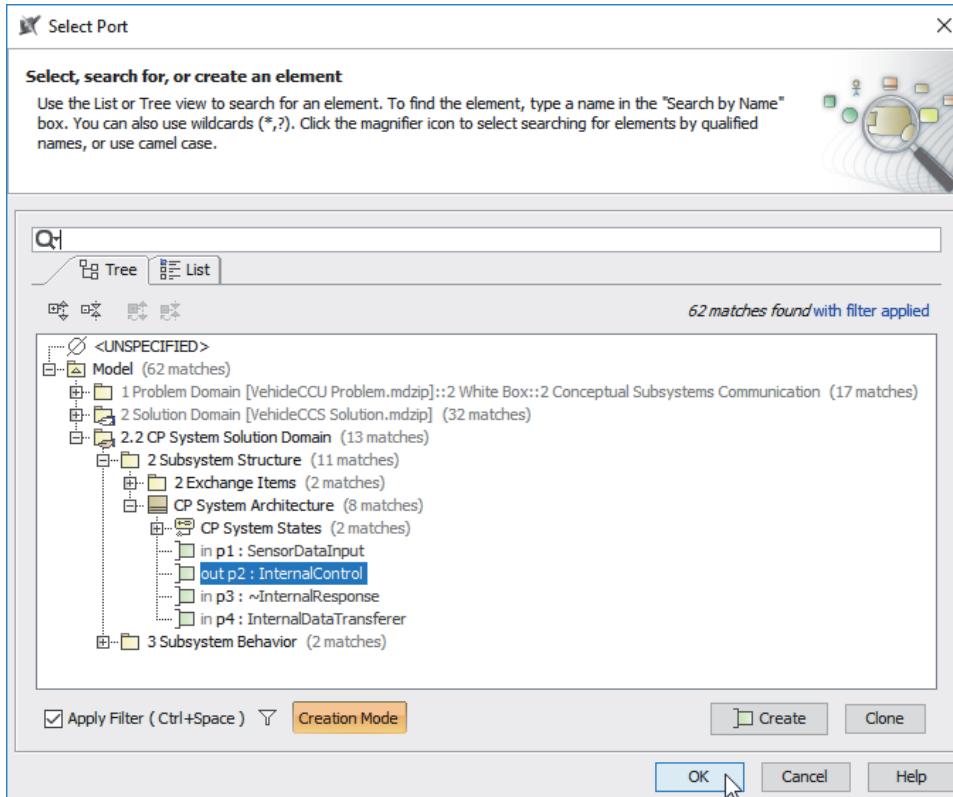


3. Click the **Initial Node** button on the diagram palette and then an empty place on the diagram pane. The initial node is created and displayed on the diagram.
4. In the Model Browser, select the *Start* signal:
 - a. Press Ctr + Alt + F.
 - b. Type *start*.
 - c. When you see the *Start* signal in the search results below, press Enter.
5. Drag the selected signal to the newly created diagram pane. The *Start* send signal action is created and displayed in the diagram pane.
6. Establish a control flow between the two nodes:
 - a. Select the initial node and click the Control Flow button on its smart manipulator toolbar.
 - b. Click the *Start* send signal action. The control flow is created.

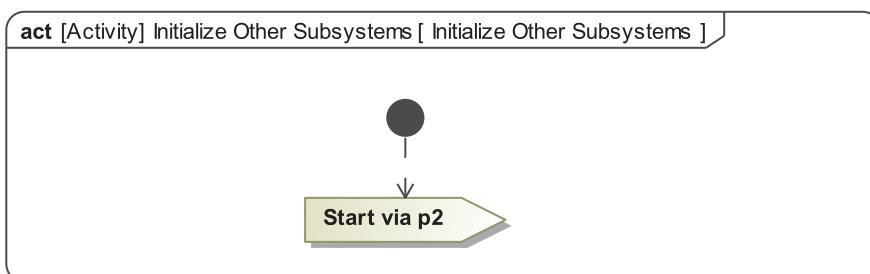


7. Select the *Start* send signal action and press Enter.

8. In the Specification of the send signal action, find the **On Port** property and select its value cell.
9. Click the button with three dots in this cell.
10. In the **Select Port** dialog, select the *p2* proxy port that belongs to the *CP System Architecture* block in the LSSA model, and close the dialog.



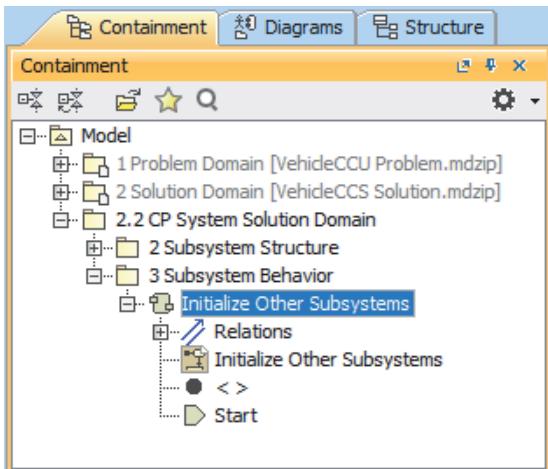
As a result, the selected proxy port name displays on the shape of the *Start* send signal action.



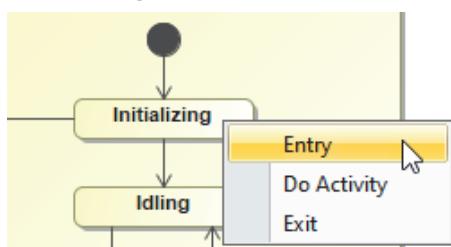
11. Open the *CP System States* diagram.

12. Select the *Initialize Other Subsystems* activity in the Model Browser and drag it onto the *Initializing* state.

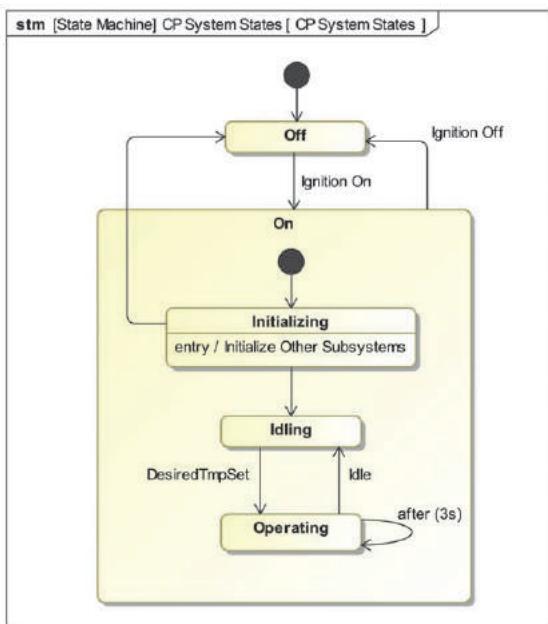
i Be sure, you selected the SysML activity, not the diagram.



13. Select **Entry**.

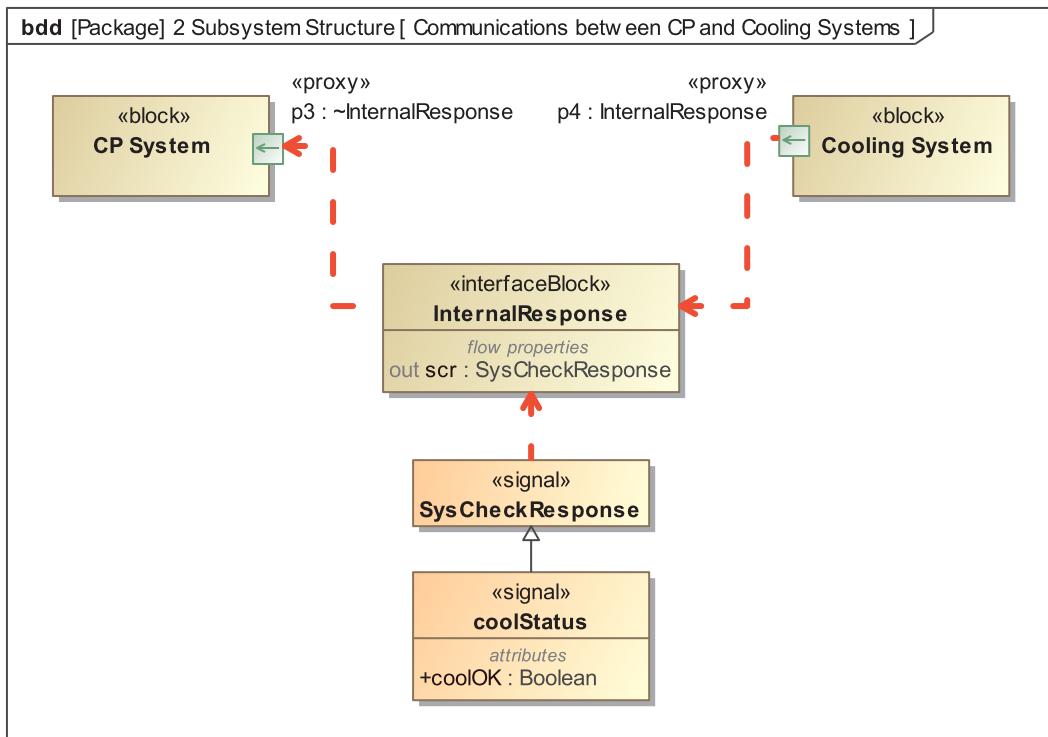


As a result, the *Initialize Other Subsystems* activity is set as the *entry* behavior of the *Initializing* state.



The engineering team appointed to build the LSSA of the Cooling System has to specify the reaction of that subsystem to the *Start* signal. Let's say the Cooling System in response sends the *coolStatus* signal with the *coolOK* attribute value *true*, if it is ready to start, and *false*, if it is not. As you can see in the following figure, the response signal is sent over the *p4* proxy port. Reactions of other logical subsystems

of the VCCS should be adequate, but we are not going to model their behavior. It's enough to have two subsystem models to learn how to model the subsystems communication.



To specify that the Cooling System, when in the *Initializing* state, performs a self-check and sends out the *coolStatus* signal with appropriate attribute value

1. Open the LSSA model of the Cooling System (the one you created in [step 1](#) of the [Subsystem Requirements](#) tutorial), if not opened yet.
2. Create a new SysML activity diagram:
 - a. In the Model Browser, select the *3 Subsystem Behavior* package.
 - b. Right-click the package and select **Create Diagram**.
 - c. 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.
 - d. Type *Perform System Check* 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.
3. Click the **Initial Node** button on the diagram palette and then an empty place on the diagram pane. The initial node is created and displayed on the diagram.

4. Create a decision node:

- a. Click the Control Flow button  on the smart manipulator toolbar of the newly created initial node.
- b. Right-click a free space below and select **Decision**.

5. Create an opaque action to set the *coolOK* attribute value to *true*:

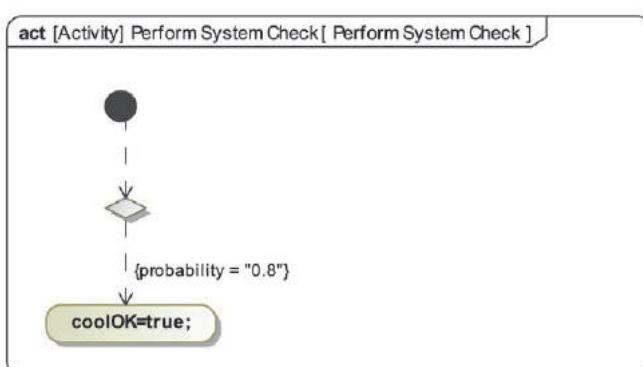
- a. Click the small black triangle next to the **Action** button on the diagram palette and then select **Opaque Action**.
- b. Click an empty place on the diagram pane. A new opaque action is created and displayed on the diagram.
- c. Type *coolOK=true;* directly on the shape.
- d. Click somewhere on the diagram pane when you're done.

6. Establish a control flow between the decision node and the newly created opaque action, with 80 percent probability:

- a. Select the decision node and click the Control Flow button  on its smart manipulator toolbar.
- b. Click the opaque action. The control flow between actions is created.
- c. Make sure the control flow is still selected and press Enter.
- d. In the Specification of the control flow, find the **Probability** property and type *0.8* in its value cell.

Control Flow	
Name	
Owner	 Perform System Check [2.1 Cooling System Solution ...
Applied Stereotype	 Probability [ActivityEdge, ParameterSet] [SysML::Activ ...
Source	 [2.1 Cooling System Solution Domain::3 Subsystem ...
Target	 coolOK=true; [2.1 Cooling System Solution Domain::...]
Guard	
To Do	
Probability	0.8
Rate	
Redefined Edge	

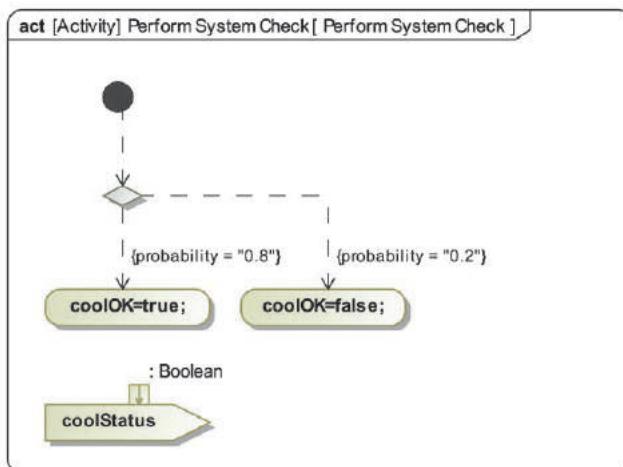
- e. Close the dialog. The probability displays on the diagram.



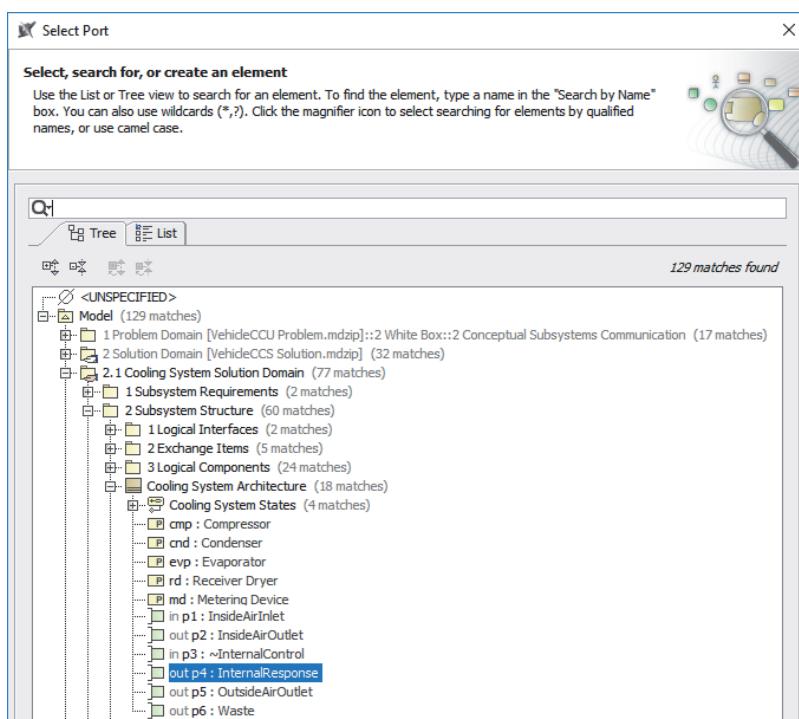
7. Follow step 5 to create another opaque action. This time set the *coolOK* attribute value to *false* (see the following figure).

8. Follow step 6 to establish another control flow, this time with 20 percent probability (see the following figure).
9. Create a send signal action to send the *coolStatus* signal out:

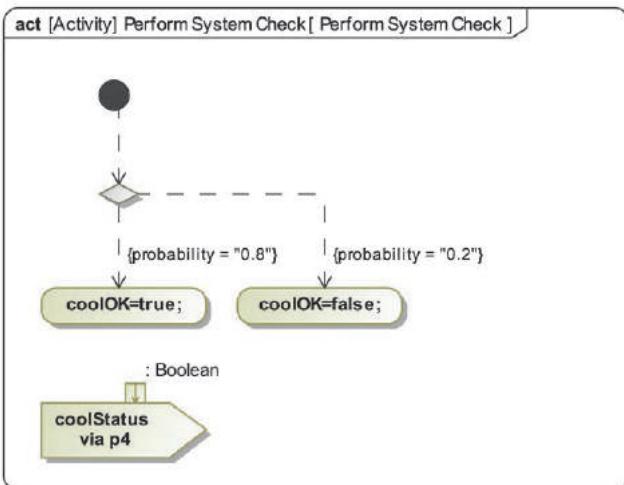
- a. In the Model Browser, select the *coolStatus* signal:
 - i. Press Ctr + Alt + F.
 - ii. Type *cools*. When you see the *coolStatus* signal in the search results below, press Enter.
- b. Drag the selected signal to the open diagram pane. The send signal action with the same name is created and displayed on the diagram pane.



- c. Keep it selected and press Enter.
- d. In the Specification window, find the **On Port** property and select its value cell.
- e. Click the button with three dots in this cell.
- f. In the **Select Port** dialog, select the *p4* proxy port that belongs to the *Cooling System Architecture* block in the LSSA model, and close the dialog.



As a result, the selected proxy port name displays on the shape of the *coolStatus* send signal action.

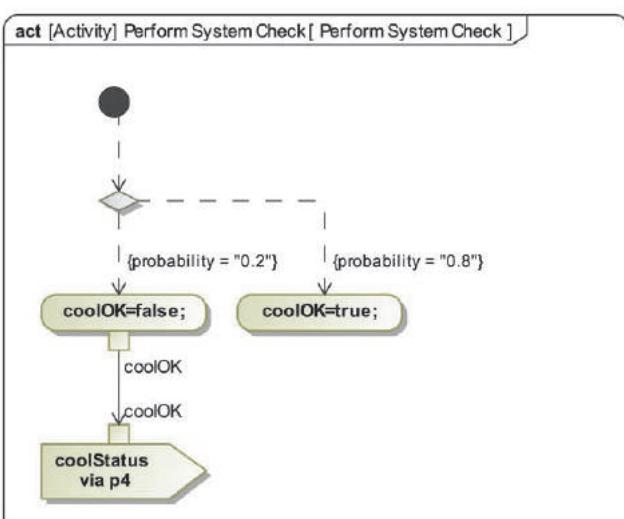


10. Establish object flows from each opaque action to the send signal action:

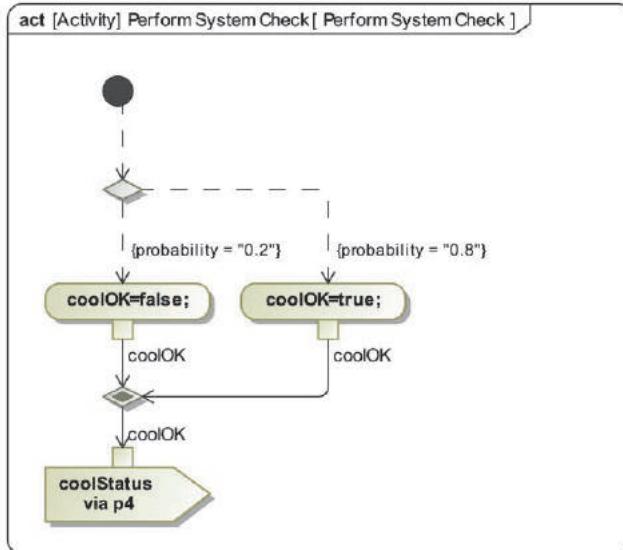
- Select one of the opaque actions on the diagram pane and then click the Object Flow button  on its smart manipulator toolbar.
- Click the input pin of the send signal action. The object flow is created, and a new output pin is attached to the opaque action.
- Click the pin label on the diagram pane. Then click it again to enter the name edit mode.
- Type *coolOK* to change the pin name.

i Note that pins do not display their types, only their names, in the following figure. For this, do the following:

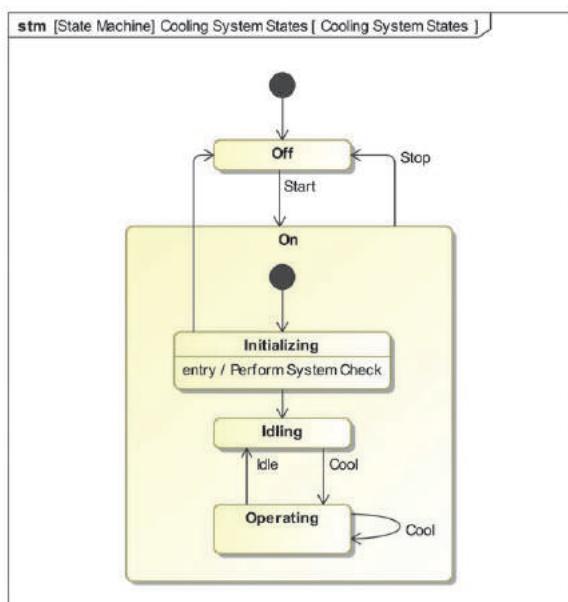
- Press down the Shift key and select both pins.
- Right-click the selection and click to select the **Show Name** check box.
- Then click **Show Type** to clear the selection.



- e. Click the **Merge** button on the diagram palette and then the newly created object flow. The merge node is created.
- f. Select another opaque action on the diagram pane and click the Object Flow button  on its smart manipulator toolbar.
- g. Click the merge node. One more object flow to the merge node is created, and a new output pin is attached to the opaque action.
- h. Click the pin label on the diagram pane. Then click it again to enter the name edit mode.
- i. Type *coolOK* to change the pin name.

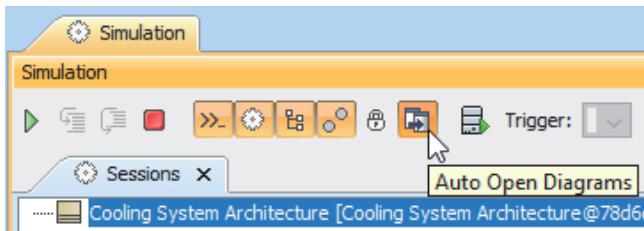


11. Open the *Cooling System States* diagram.
12. Select the *Perform System Check* activity in the Model Browser (be sure that you selected the SysML activity, not the diagram!) and drag it onto the *Initializing* state.
13. Select **Entry**. As a result, the *Perform System Check* activity is set as the *entry* behavior of the *Initializing* state.

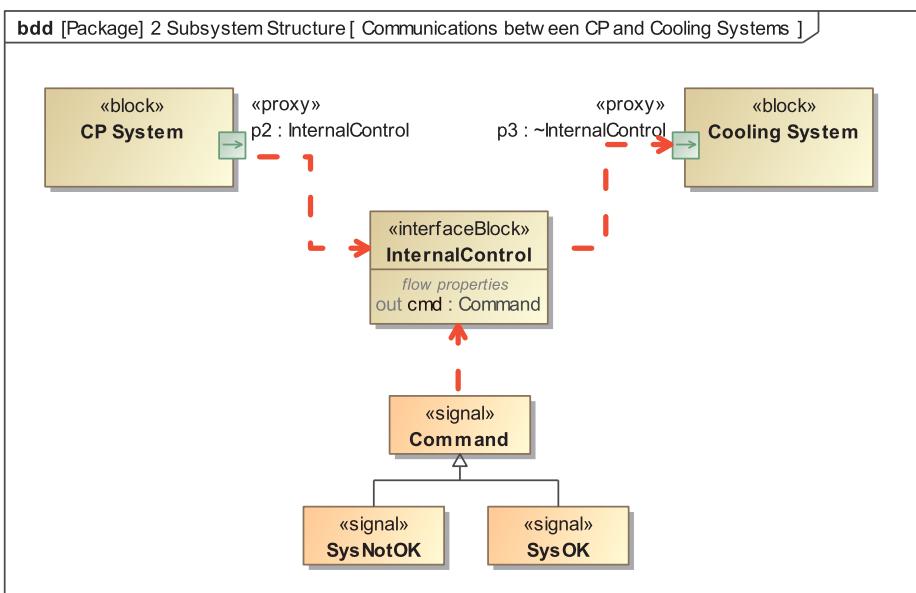


When you're done with this state machine diagram, try to run the simulation again. Before starting the model execution, be sure the Auto Open Diagrams mode is enabled (see the following figure). In this mode, the tool automatically opens the diagram that is currently executed. Hence, when the model execution switches from the *Cooling System States* diagram to the *Perform System Check* diagram (the one

that specifies the *entry* behavior of the *Initializing* state), the *Perform System Check* diagram is opened automatically.



The appointed engineering team also has to specify what the CP System does after it gets the responses from other subsystems. It should evaluate the status of all subsystems according to this information and send them signals triggering them to move either to the *Idling*, or to the *Off* state. As you can see in the following figure, the appropriate signals are sent over the *p2* proxy port.

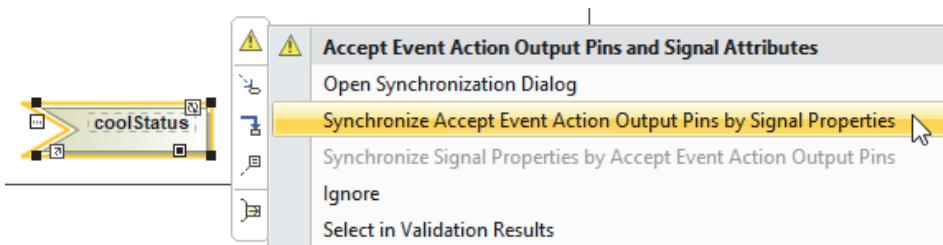


Note that the *SysOK* and *SysNotOK* signals are not defined in any of your models, although they should have been identified in the LSA model by the system architect many cells ago (see [step 5 of the System Structure tutorial](#)). This means that the system architect must be informed about two new identified signals. As soon as he/she updates the LSA model, the signals can be used in the LSSA models of all logical subsystems. Here we expect you to update the LSA model appropriately on your own.

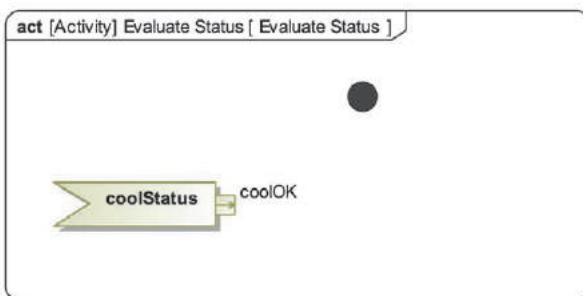
To specify that the CP System evaluates the status of all subsystems and sends them appropriate signals

1. Open the LSSA model of the CP System (the one you created in the previous step), if not opened yet.
2. Create a new SysML activity diagram:
 - a. In the Model Browser, select the *3 Subsystem Behavior* package.
 - b. Right-click the package and select Create Diagram.
 - c. 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.

- d. Type *Evaluate Status* 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.
3. Click the **Initial Node** button on the diagram palette and then an empty place on the diagram pane. The initial node is created and displayed on the diagram.
4. Create an accept event action waiting for the *coolStatus* signal:
- a. Click the **Accept Event Action** button on the diagram palette and then an empty place on the diagram pane. The accept event action is created and displayed on the diagram.
 - b. In the Model Browser, select the *coolStatus* signal:
 - i. Press Ctr + Alt + F.
 - ii. Type *cools*.
 - iii. When you see the *coolStatus* signal in the search results below, press Enter.
 - c. Drag the selected signal onto the shape of the accept event action. It becomes the *coolStatus* accept event action.
 - d. Double-click the accept event action.
 - e. In its Specification, find the **Is Unmarshall** property and set its value to *true*.
 - f. Close the dialog.
 - g. When the shape of the *coolStatus* accept event action is surrounded by the yellow border, select the shape and click the button with the yellow triangle (see the following figure).
 - h. Select **Synchronize Accept Event Action Output Pins by Signal Properties**.



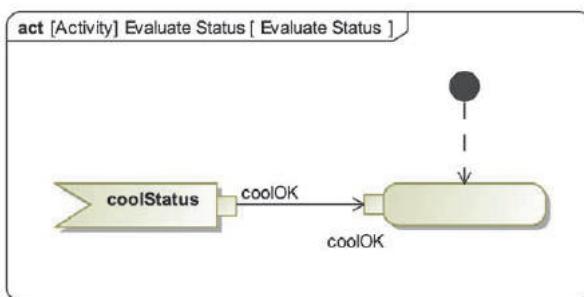
The output pin named and typed after the attribute of the *coolStatus* signal is attached to the shape of the accept event action.



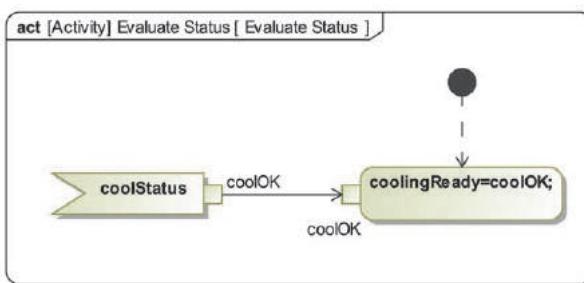
5. Create an empty opaque action:
- a. Click the small black triangle next to the **Action** button on the diagram palette and then select **Opaque Action**.

- b. Click an empty place on the diagram pane. A new opaque action is created and displayed on the diagram.
 - c. Click somewhere on the diagram pane when you're done.
6. Establish a control flow between the initial node and the opaque action:
- a. Select the initial node and click the Control Flow button  on its smart manipulator toolbar.
 - b. Click the opaque action. The control flow is created.
7. Establish an object flow between the *coolStatus* accept event action and the opaque action:

- a. Select the output pin of the *coolStatus* accept event action and click the Object Flow button  on its smart manipulator toolbar.
- b. Click the opaque action. The object flow is created along with the input pin attached to the opaque action.
- c. Click the label of the input pin on the diagram pane. Then click it again to enter the name edit mode.
- d. Type *coolOK* to rename the pin and press Enter.



8. Specify the opaque action expression:
- a. Select the shape of the opaque action.
 - b. Type *coolingReady=coolOK;* directly on the shape.
 - c. Click somewhere on the diagram pane, when you're done.



9. Create the *coolingReady* value property for the *CP System Architecture* block:
- a. In the Model Browser, right-click the block and then select **Create Element**.
 - b. In the search box, type *vp*, the acronym of the value property, and press Enter. The new value property is created and appears in the name edit mode.

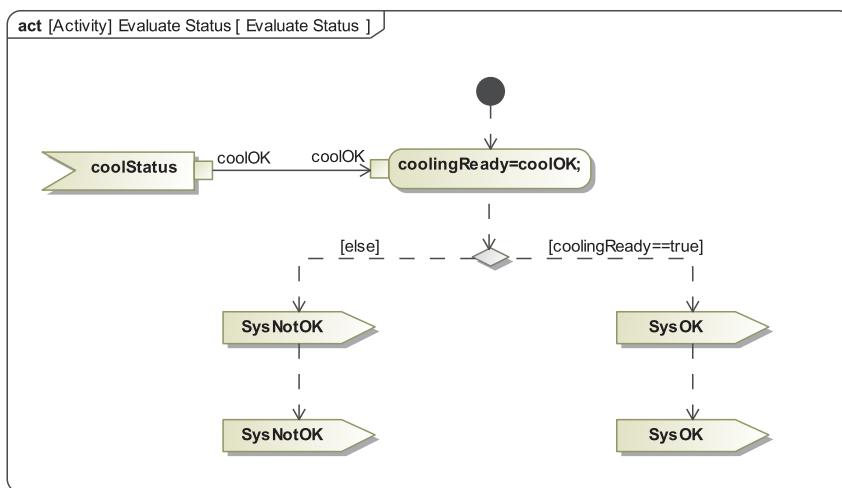
- c. Type `coolingReady:Bool` and press Ctrl + Spacebar.
 - d. When you see the `Boolean` type in the list below, select it and press Enter.
10. Create a decision node:
- a. Click the Control Flow button  on the smart manipulator toolbar of the opaque action.
 - b. Right-click a free space below and select **Decision**.
11. Create two of the `SysOK` send signal actions:
- a. In the Model Browser, select the `SysOK` signal:
 - i. Press Ctr + Alt + F.
 - ii. Type `sys0`.
 - iii. When you see the `SysOK` signal in the search results below, press Enter.
 - b. Drag the selected signal to the diagram pane twice. Two of the `SysOK` send signal actions are created and displayed in the diagram pane.
12. Repeat previous step to create two more of the `SysNotOK` send signal actions.
13. Create control flows you see in the following figure on your own.
14. Specify the condition when all the subsystems transit to the `Idling` state:

- a. Select the control flow on the right.
- b. Type `[coolingReady==true]` and press Enter. The guard is specified.

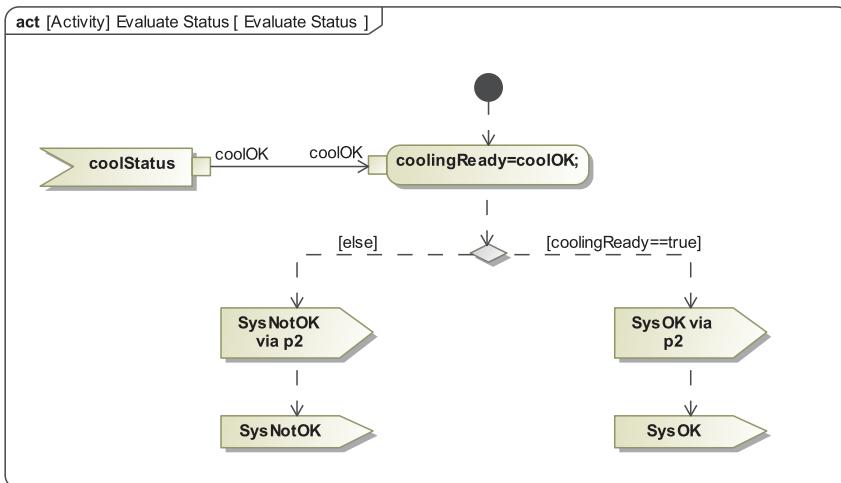
 Note that the closing bracket is added automatically. You don't need to do this.

15. Specify the condition when all the subsystems transit to the `Off` state:

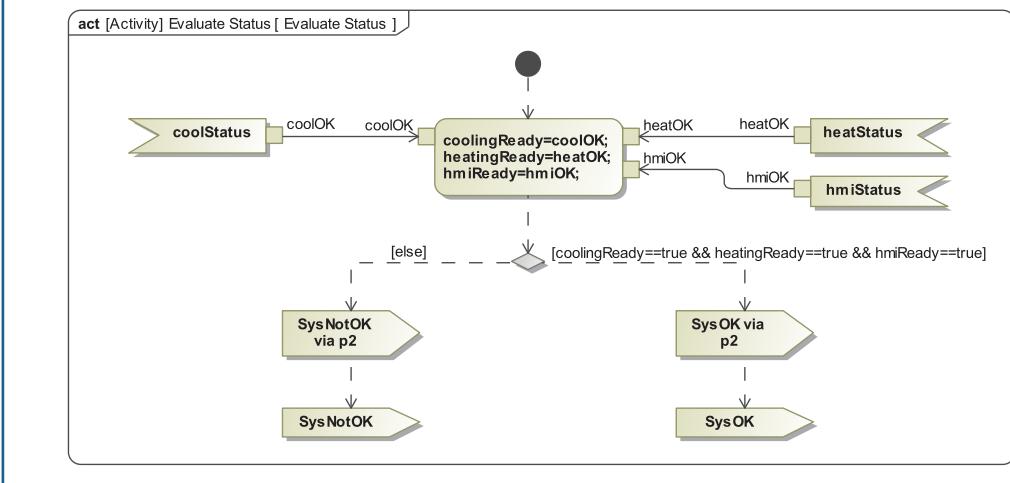
- a. Select the control flow on the left.
- b. Type `[else]` and press Enter.



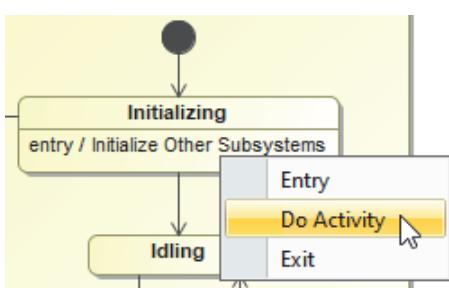
16. Specify that one of the *SysOK* send signal actions is sent via the *p2* proxy port of the *CP System Architecture* block on your own.
17. Specify that one of the *SysNotOK* send signal actions is sent via the *p2* proxy port of the *CP System Architecture* block on your own.



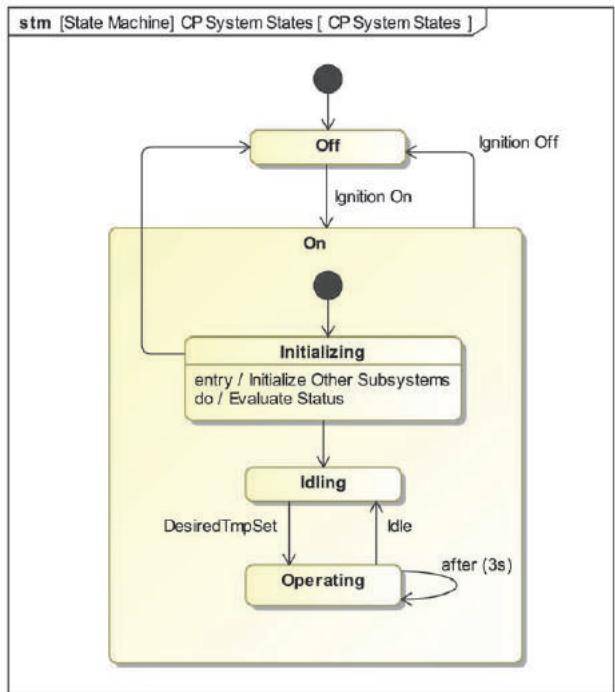
i The following diagram shows how the *Evaluate Status* diagram would look, if you specified that the CP System takes into consideration the status of all logical subsystems.



18. Open the *CP System States* diagram.
19. Select the *Evaluate Status* activity in the Model Browser and drag it onto the *Initializing* state.
20. Select **Do Activity**.

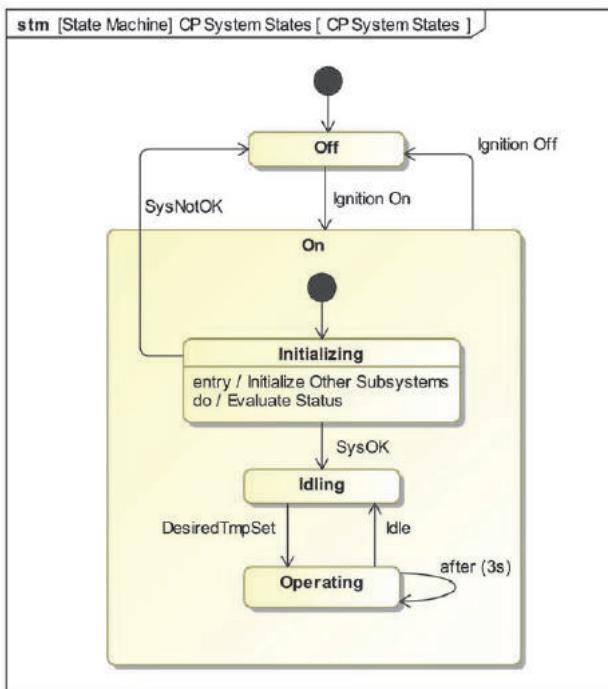


As a result, the *Evaluate Status* activity is set as the *do* behavior of the *Initializing* state.

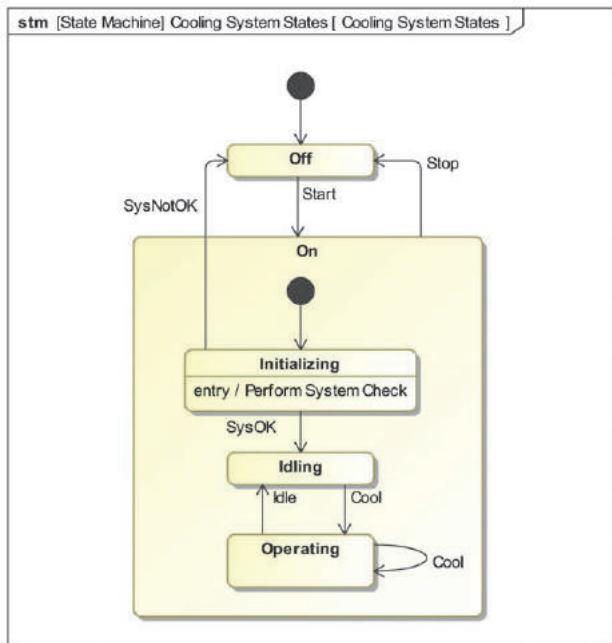


Afterwards, state machine diagrams of both logical subsystems must be updated, by specifying the signals which trigger the transitions to the *Idling* and *Off* states. You can do this on your own (see [step 4 of this cell tutorial](#), if you need to refresh your knowledge).

This is the final view of the *CP System States* diagram.

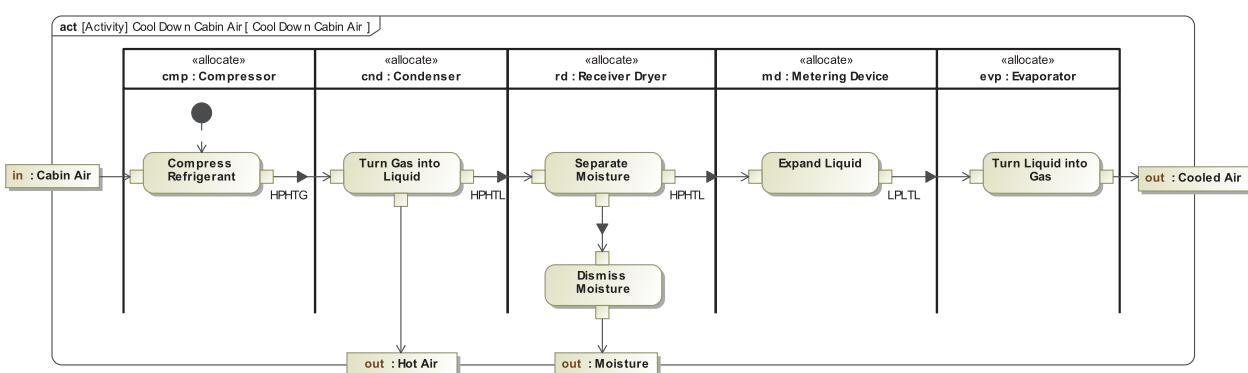


And here is the final view of the *Cooling System States* diagram.

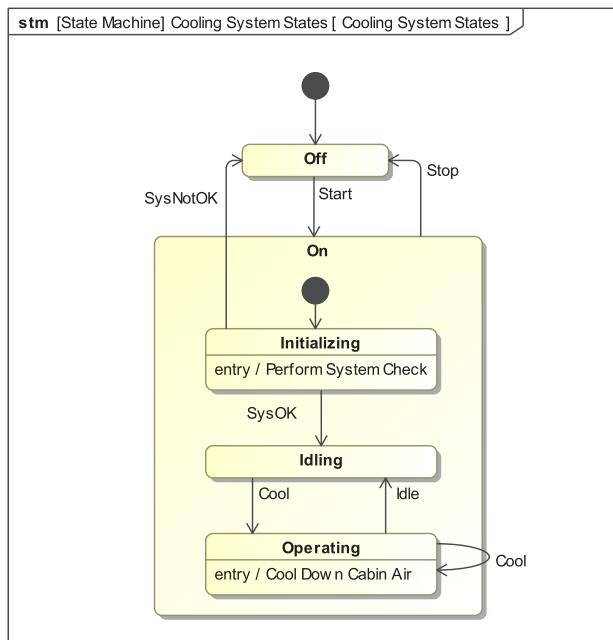


Step 6. Synchronizing item flows from the structure model

As in the problem domain model, the item flows you identified in the structure model of the logical subsystem architecture can also be displayed in its behavior model. The following figure shows the item flows in the *Cool Down Cabin Air* activity diagram (the *HPHTG* and *HPHTL* signals, among others) are synchronized from the *Cooling System Logical Components* ibd you created in [step 4](#) of the [Subsystem Structure](#) tutorial. Remember, this can be easily done after you allocate actions to appropriate swimlanes.

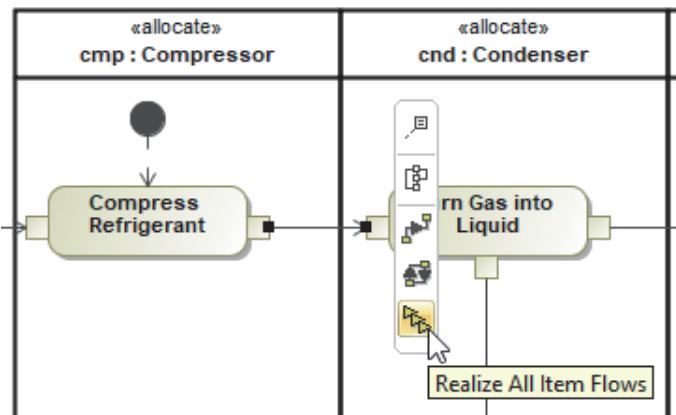


The *Cool Down Cabin Air* diagram defines the behavior of the Cooling System when it is in the *Operating* state (see the following figure). As you already know how to create a SysML activity diagram and set the selected SysML activity as an *entry*, *do*, or *exit* behavior of the state, it is not explained here how to do this. However, the depicted diagrams are available in the *Cooling System Solution* model, which can be found in <the installation folder of the **modeling tool**>\samples\MagicGrid (i.e., C:\Program Files\ Cameo Systems Modeler 2021x\samples\MagicGrid).

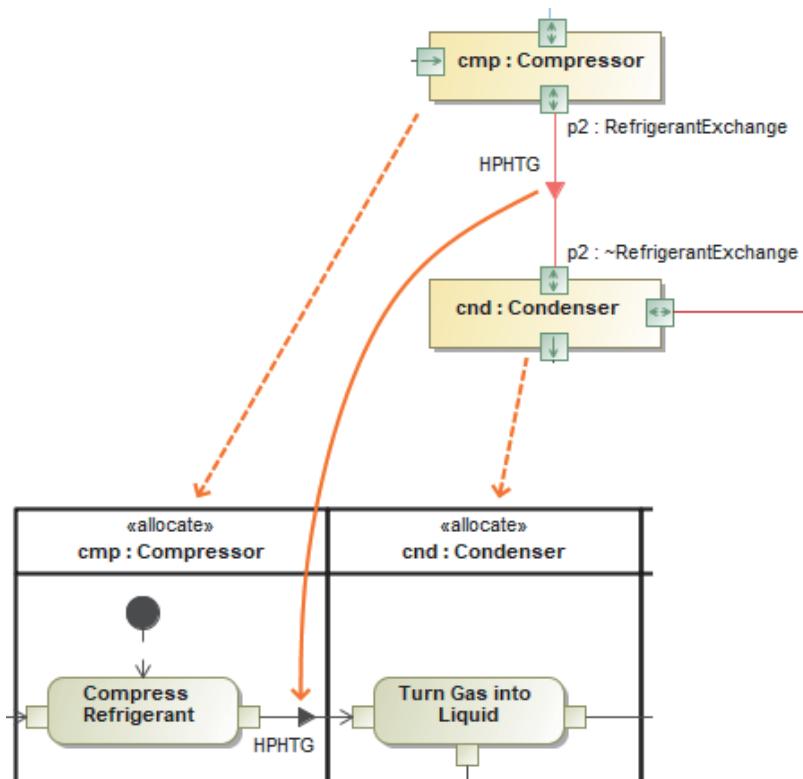


To synchronize the item flow from the corresponding ibd

1. Select the object flow between the *Compress Refrigerant* and *Turn Gas Into Liquid* actions.
2. Click the Realize All Item Flows button on the smart manipulator toolbar of the object flow.



The item flow is synchronized from the *Cooling System Logical Components* ibd and displayed on the object flow.



Subsystem behavior done. What's next?

- Once you define the supposed behaviors of all the logical subsystems, you can start integrating their LSSA models into the whole and create diverse configurations of the **Sol** model (see Chapter [Building system configuration model](#)).
- You can also go to the **Subsystem Parameters** cell and determine the method for calculating one or more subsystem parameters.

Subsystem Parameters

What is it?

You can go to this cell as soon as the subsystem structure is captured in the model.

Subsystem parameters capture quantitative characteristics of the relevant logical subsystem, that you can measure. They are used to calculate MoEs for that logical subsystem, identified during the problem domain analysis (see Chapter [MoEs for Subsystems](#)). MoEs for subsystems captured in the solution domain model satisfy non-functional quantitative subsystem requirements.

Mathematical expressions for MoEs derivation from subsystem parameters should also be specified in this cell. Knowing how to calculate MoEs enables early subsystem 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.

It's important to note here that not all MoEs in the solution domain must be based on the MoEs in the problem domain model. Some of them can be identified only when building the logical architecture of the logical subsystem.

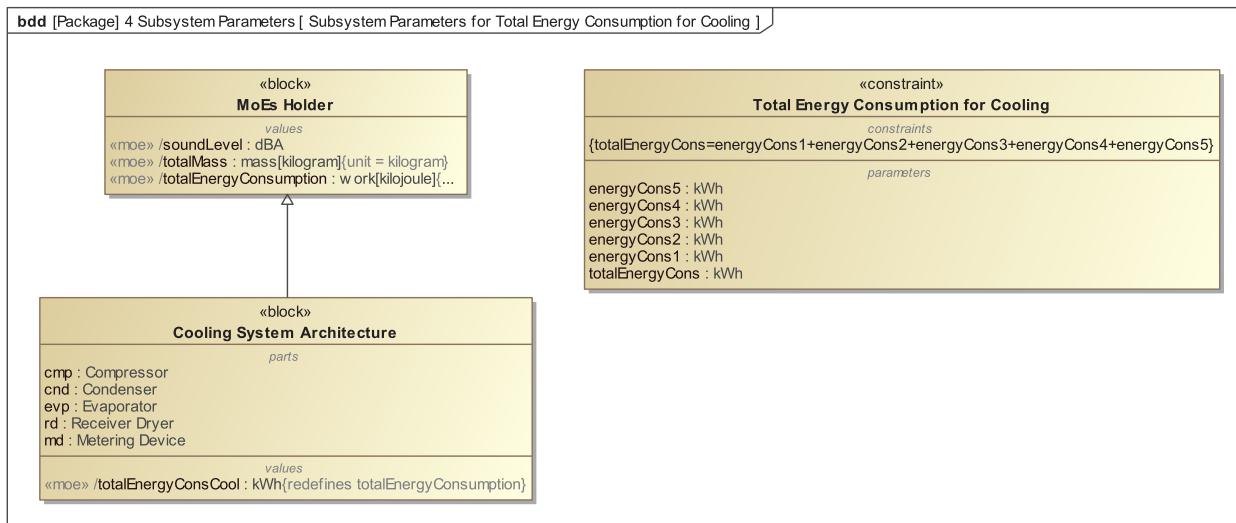
Who is responsible?

Parameters of the appointed logical subsystem can be captured by the Systems Architect or Systems Engineer, who belongs to the engineering team responsible for building the LSSA of that logical subsystem.

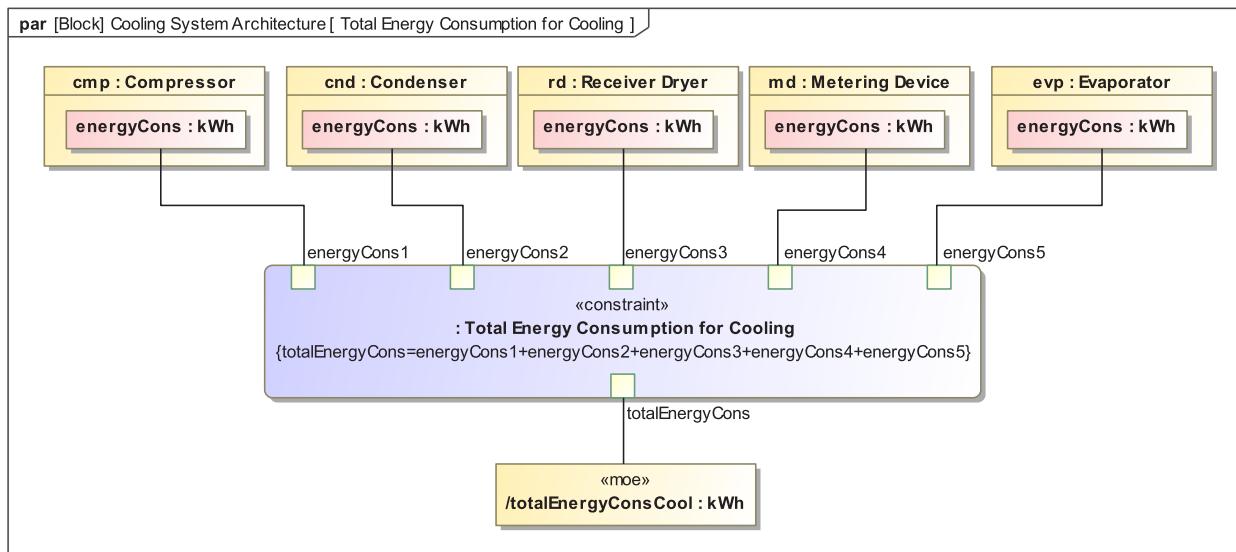
How to model?

Just like MoEs for the whole **system of interest**, the MoEs for logical subsystems can be captured in the model as value properties with the «moe» stereotype applied. These value properties must belong to the block, which represents the relevant logical subsystem. You must also know that measures of effectiveness can be inherited from the reusable sets of MoEs, so called MoEs holders, instead of specifying them from scratch.

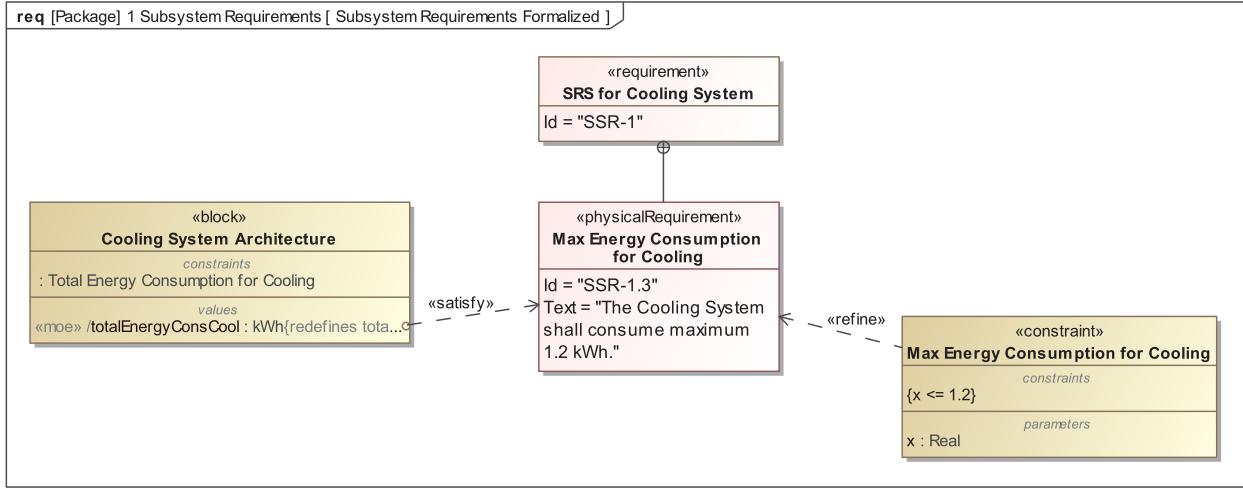
The formula for calculating the MoE can be captured as a constraint expression of some constraint block. The infrastructure of the SysML block definition diagram can be utilized for capturing MoEs, constraint expressions, and even required subsystem parameters. The following figure displays the bdd that captures the *totalEnergyConsCool* measure of effectiveness and the formula for calculating this MoE as the constraint expression of the *Total Energy Consumption for Cooling* constraint block.



To empower the constraint expression to perform calculations, you have to specify what subsystem parameters should be consumed as variables (or, in terms of SysML, constraint parameters) of that constraint expression. Just like MoEs, subsystem parameters can be captured in the model as value properties of the block that captures the logical subsystem or any of its components, only without any stereotype. The SysML parametrics diagram can be used to relate the subsystem parameters as well as MoEs to the constraint parameters: former as inputs, latter as outputs. Binding connectors represented as solid lines in the following figure, should be used for this.



The simulation capabilities of the **modeling tool** allow you to calculate MoEs with given inputs. Having the MoE calculated, you can verify the appropriate non-functional subsystem requirement and show 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 create a satisfy relationship from the value property, which captures the MoE, to the appropriate subsystem requirement. The following figure displays an example of the satisfy relationship between the *totalEnergyConsCool* value property and the *Max Energy Consumption for Cooling* non-functional requirement, which means that the requirement is satisfied when the value of the *totalEnergyConsCool* value property is not greater than 1.2 kWh. As you can see, the natural language expression in the requirement text is refined by another constraint expression of the *Max Energy Consumption for Cooling* constraint block with the $x \leq 1.2$ inequality.



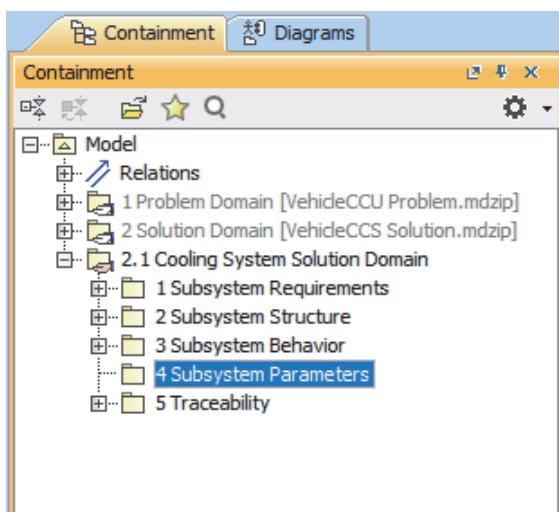
Both satisfy and refine relationships are represented in the SysML requirements diagram, but the bdd can also be used for this purpose.

Tutorial

- Step 1. Organizing the model for subsystem parameters
- Step 2. Specifying the MoE for capturing the total energy consumption for cooling
- Step 3. Defining a method to calculate the total energy consumption for cooling
- Step 4. Specifying subsystem parameters for calculating the total energy consumption for cooling
- Step 5. Binding constraint parameters to appropriate value properties
- Step 6. Performing early subsystem requirements verification
- Step 7. Recalculating subsystem parameters in different states

Step 1. Organizing the model for subsystem parameters

The constraint blocks that capture expressions for calculating subsystem parameters can be stored in a separate package in the 2.1 *Cooling System Solution Domain* package. Following the layout of the MagicGrid framework, its name must begin with number 4.



To organize the model for the subsystem parameters

1. Open the solution domain model of the Cooling System (the *Cooling System Solution.mdzip* file) you created in **step 1** of the **Subsystem Requirements** 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 *4 Subsystem Parameters* to specify the name of the new package and press Enter.

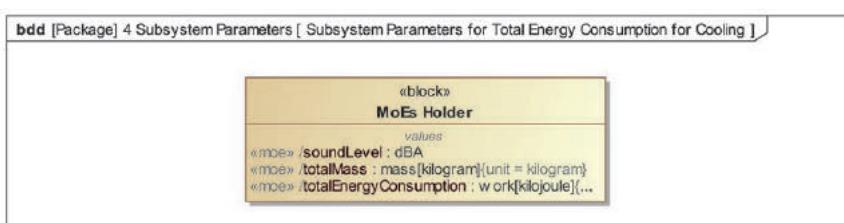
Step 2. Specifying the MoE for capturing the total energy consumption for cooling

According to the subsystem requirements specification (see Chapter **Subsystem Requirements**) and the MoE identified for the *Cooling* block in the problem domain model (see **MoEs for Subsystems**), the appointed engineering team must specify the MoE for capturing the total energy consumed by the Cooling System.

It was already mentioned that MoEs can be specified from scratch or inherited form the MoEs holder. The following procedure describes the latter case. The inherited MoE is redefined to change its name and type. Redefinition is necessary for automated requirements verification, which is performed in one of the following steps.

To specify the MoE for capturing the total energy consumed by the Cooling System

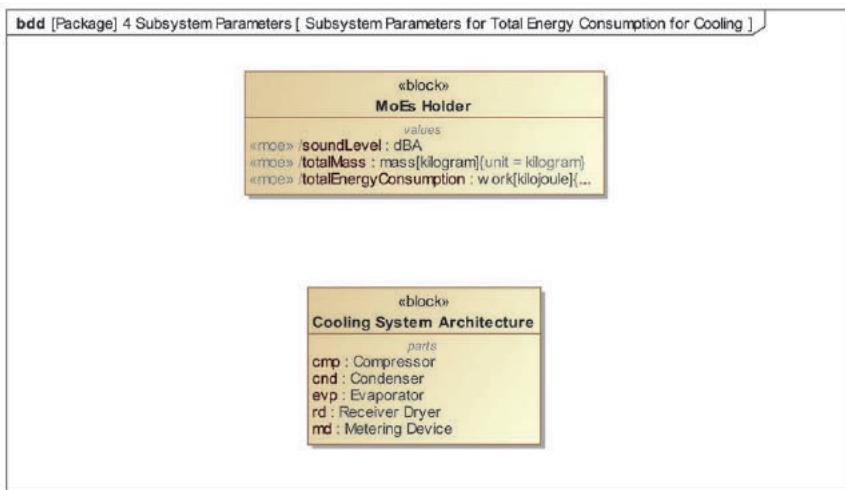
1. Create a bdd for capturing system parameters:
 - a. Right-click on the *4 System Parameters* package you created in previous step and select **Create Diagram**.
 - b. In the search box, type *bdd* (the acronym of the SysML block definition diagram), and then double-press Enter. The diagram is created.
 - c. Type *Subsystem Parameters for Total Energy Consumption for Cooling* to specify the name of the new diagram and press Enter again.
2. Display the *MoEs Holder* block on the diagram:
 - a. Press Ctrl + Alt + F to open the **Quick Find** dialog.
 - b. In the dialog, type *moes*.
 - 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.



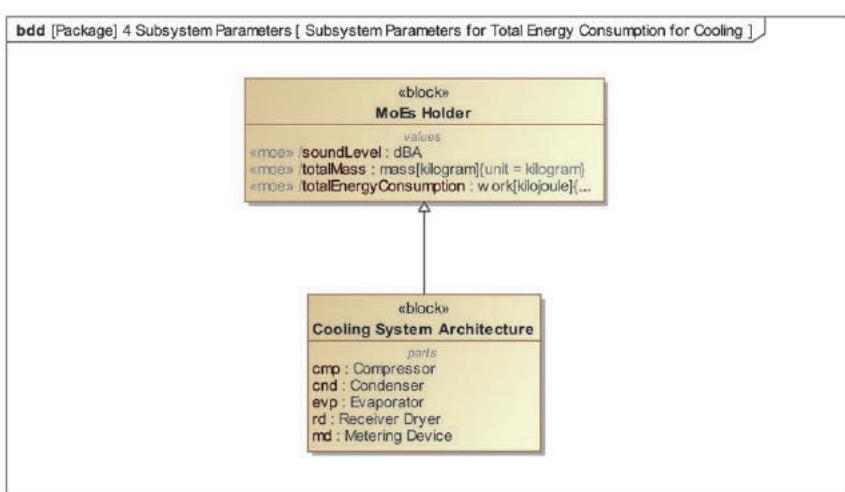
3. Display the *Cooling System Architecture* block on the diagram:

- 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 *cooling system a*.
- c. When you see the *Cooling System Architecture* block selected in the search results list, press Enter. The block is selected in the Model Browser.
- d. Drag the *Cooling System Architecture* block to the diagram pane. The shape of the block is displayed on the diagram.

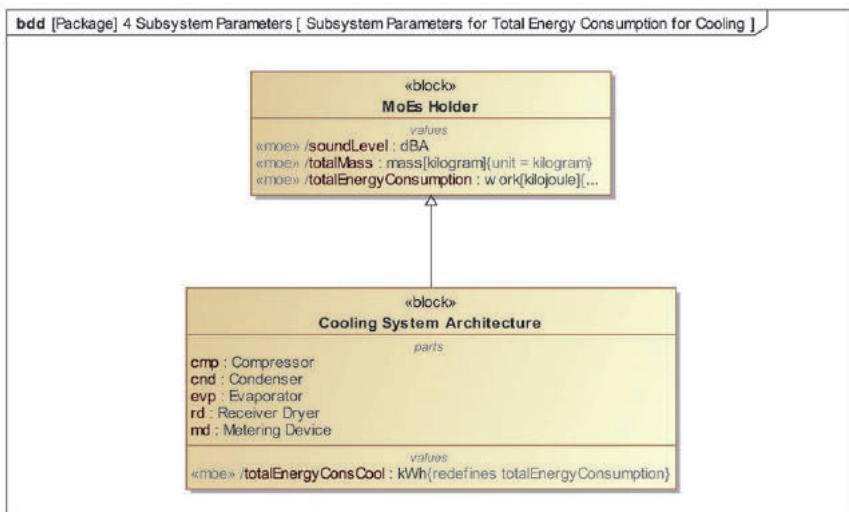


- 4. Establish the generalization between the *Cooling System Architecture* block and the *MoEs Holder* block:
 - a. Select the shape of the *Cooling System Architecture* block and click the Generalization button on its smart manipulator toolbar.
 - b. Select the shape of the *MoEs Holder* block. The *MoEs Holder* block becomes the super-type of the *Cooling System Architecture* block, which means that the latter inherits all *MoEs* from the former.



- 5. Redefine the *totalEnergyConsumption* value property for the *Cooling System Architecture* block:
 - a. Right-click the *Cooling System Architecture* block and select **Specification**.
 - b. On the left of the open dialog, click **Properties**.
 - c. On the right, under the **Value Properties** category, select the *totalEnergyConsumption* value property.
 - d. Click the **Redefine** button.
 - e. In the **Type** cell, immediately type *kWh* and press Enter. The *kWh* value type is created in the model and set as type for the redefined value property.

- f. In the **Name** cell, type *totalEnergyConsCool* to overwrite the current property name and press Enter. The inherited value property under this new name now belongs to the *Cooling System Architecture* block.
- g. Click **Close**.



i As you can see, the inherited value property it redefines displays in the curly brackets.

Step 3. Defining a method to calculate the total energy consumption for cooling

The next thing you need to do is capture the mathematical expression which specifies how to calculate the MoE, minutes ago specified for the Cooling System. It can be captured as the constraint expression of some constraint block stored in your model.

Let's say the total energy consumption of the Cooling System equals a sum of energies consumed by all its components. The constraint expression can be defined as follows:

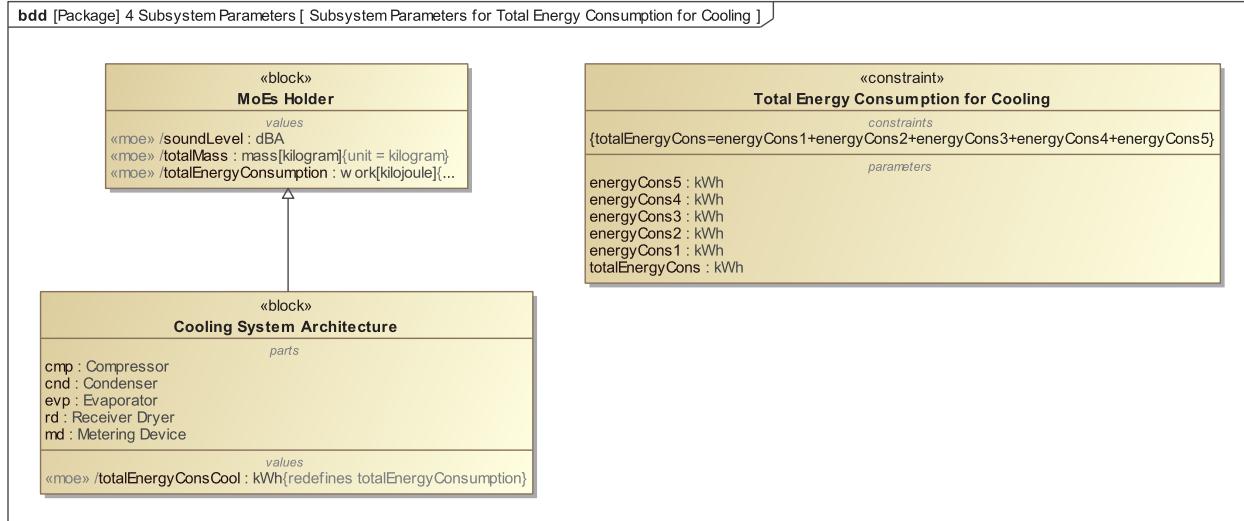
totalEnergyCons = energyCons1 + energyCons2 + energyCons3 + energyCons4 + energyCons5

where:

1. *totalEnergyCons* stands for the total energy consumed by the Cooling System
2. *energyCons1* may stand for the energy consumed by the Compressor
3. *energyCons2* may stand for the energy consumed by the Condenser
4. *energyCons3* may stand for the energy consumed by the Receiver Dryer
5. *energyCons4* may stand for the energy consumed by the Metering Device
6. *energyCons5* may stand for the energy consumed by the Evaporator

In terms of SysML, *totalEnergyCons*, *energyCons1*, ..., and *energyCons5* 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 created in [step 2 of this cell tutorial](#). The procedure of capturing the method for calculating the total energy consumption of the Cooling System is adequate to the one described in [step 3 of System Parameters tutorial](#). Once you capture the method, the *Subsystem Parameters for Total Energy Consumption for Cooling* bdd should look like in the following figure.

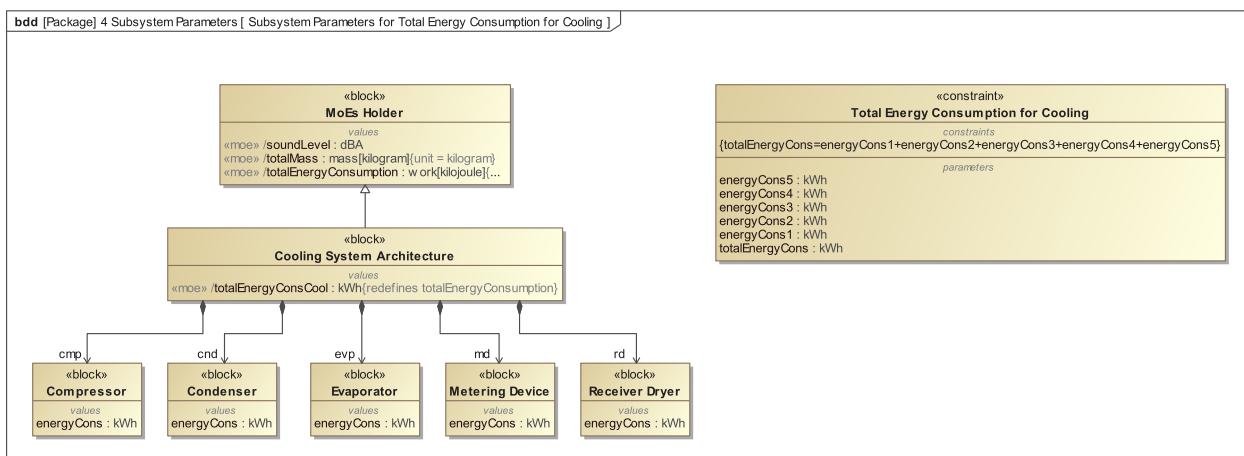


Step 4. Specifying subsystem parameters for calculating the total energy consumption for cooling

Having the constraint expression, you can easily determine what subsystem or, as in this case, component parameters are necessary for calculating the total energy consumed by the Cooling System. Input parameters of the *Total Energy Consumption for Cooling* constraint block can help with this task.

The *Total Energy Consumption for Cooling* constraint block has five input parameters, each for the energy consumption of a single component of the Cooling System, and therefore requires five component parameters in your model. Appropriate value properties should be specified for each block that captures one of the components of the Cooling System.

The procedure for capturing the above mentioned component parameters is adequate to the one described in [step 4 of System Parameters tutorial](#). Once you're done with this, the *Subsystem Parameters for Total Energy Consumption for Cooling* bdd should look like the following figure.

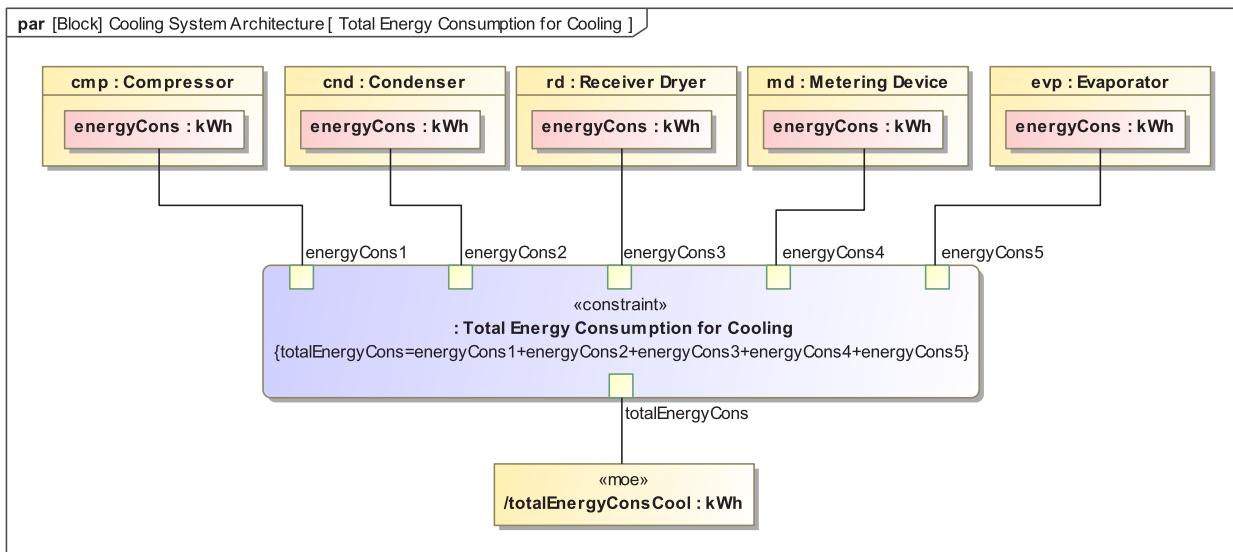


Step 5. Binding constraint parameters to appropriate value properties

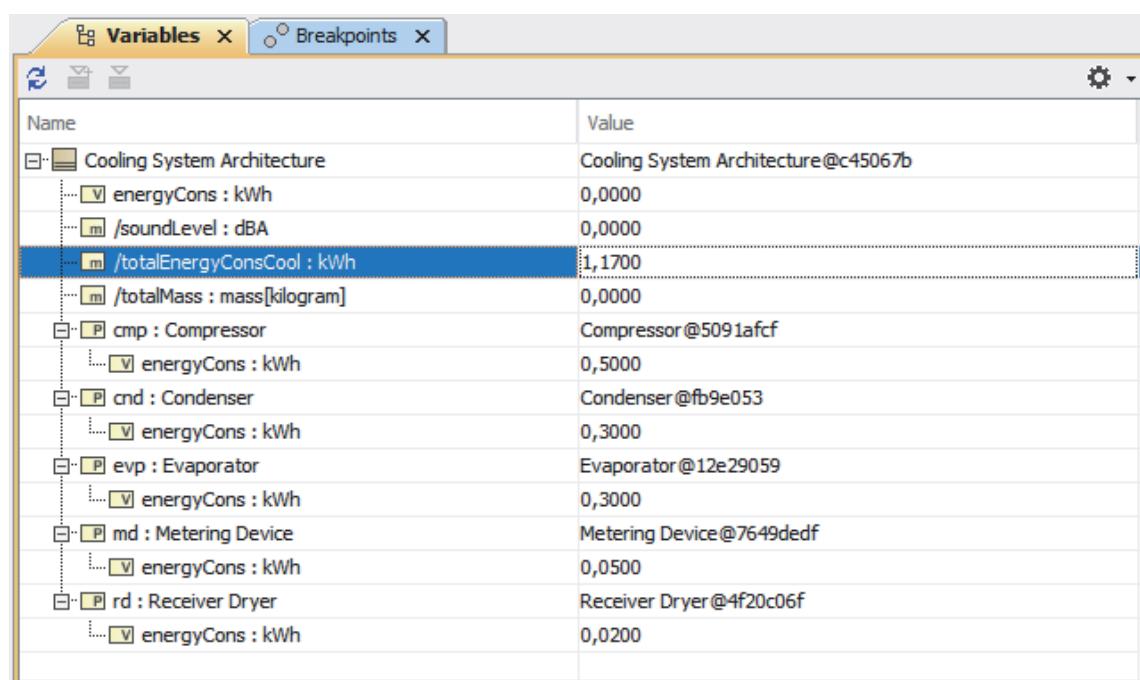
Once you have all the necessary value properties in the model, it's time to bind them to the appropriate constraint parameters. Remember that it is the binding which empowers the constraint expression to perform calculations on given inputs.

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 *Cooling System Architecture* block. However, creating the SysML parametrics diagram is not enough to get started. Establishing binding connectors is possible only after the constraint property typed by the *Total Energy Consumption for Cooling* constraint block is created for the *Cooling System Architecture* block. Be aware that the same constraint block can be used in multiple contexts, and binding connectors are context-specific relationships.

If you name the SysML parametrics diagram *Total Energy Consumption for Cooling*, the procedure of establishing the bindings is very similar to the one described in **step 5** of the **System Parameters** tutorial. Once you're finished with this, the *Total Energy Consumption for Cooling* parameters diagram in your model should look like the one below.

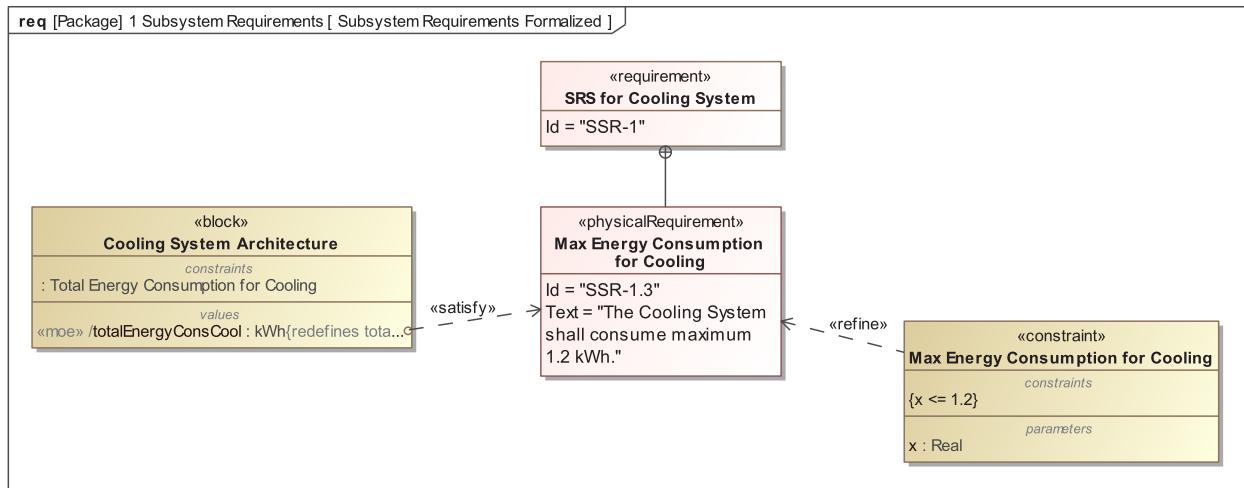


Now you can execute the model, provide input values, and see the result of the calculation. To start the execution, click the Run button in the toolbar above the *Total Energy Consumption for Cooling* diagram. When in the **Variables** tab of the **Simulation** panel (by default, it is on the right side of the **Simulation** panel), you can specify the input values and see the output value (result of the calculation).

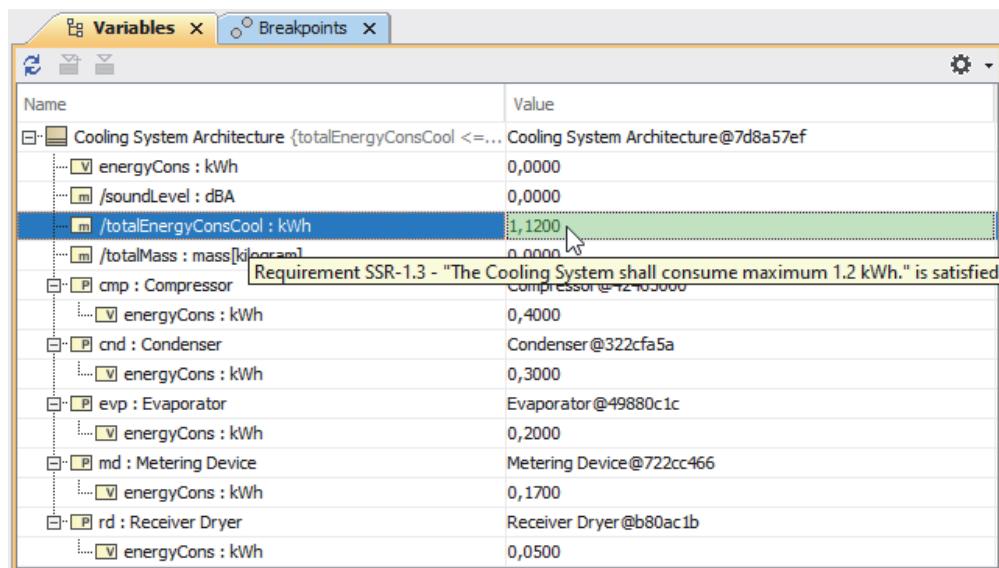


Step 6. Performing early subsystem requirements verification

Having the concrete value of the value property that captures the MoE, you can verify the related subsystem requirement and show whether it is satisfied or not. In this case, you can perform the verification of the *Max Energy Consumption for Cooling* subsystem requirement, which insists that "The Cooling System shall consume maximum 1.2 kWh". The procedure for getting ready for this verification is adequate to the one described in [step 6 of System Parameters tutorial](#). If you name the requirement diagram *Subsystem Requirements Formalized*, it should look very similar to the one below when you're done.



Now, if you go back to the *Total Energy Consumption for Cooling* diagram and execute the model again, you should see that the *totalEnergyConsCool* value property is highlighted in green, which means that the *Max Energy Consumption* system requirement is satisfied. Try to change the input values and see how the highlight color changes when the requirement is not satisfied. This is how you perform the automated requirements verification. Moreover, this way you can perform manual or automatic (it requires some algorithm to be defined) parameters optimization in order to find the optimal system configuration.



These values can be stored in the model as SysML instance specifications and their slot values. If you cannot remember how to do this, refer to [step 7 of the System Parameters tutorial](#).

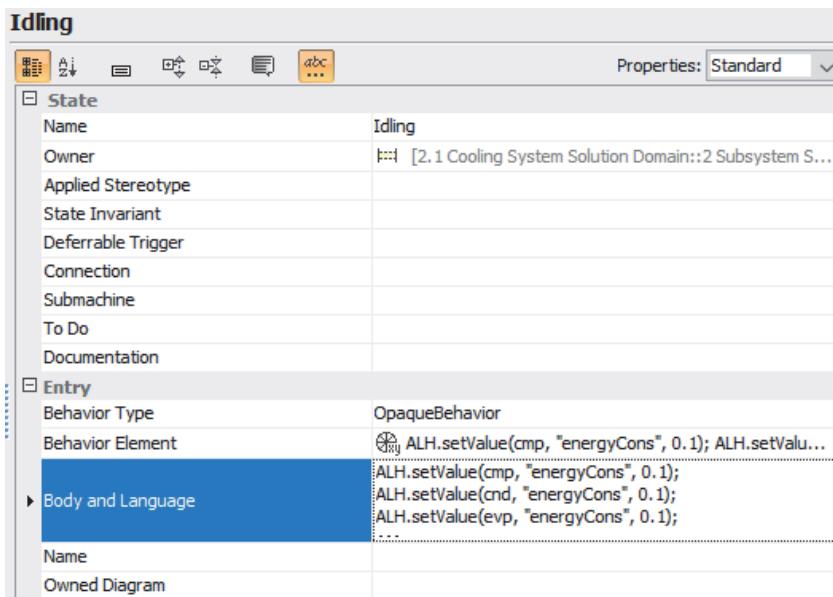
Step 7. Recalculating subsystem parameters in different states

At times, values used for calculating a subsystem parameter may be dynamic and may vary in different states of the subsystem. As a result, the system parameter values may vary, too.

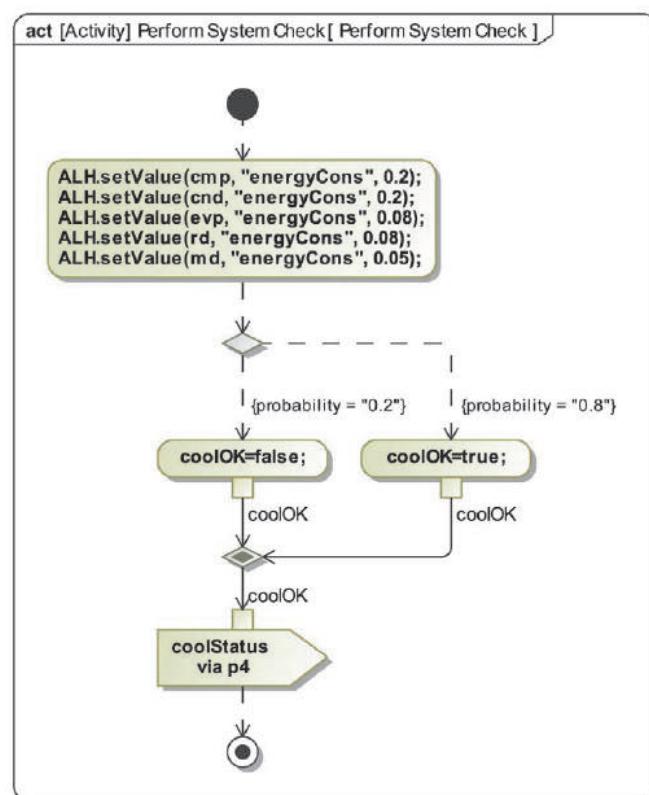
To illustrate this, let's assume that the components of the Cooling System consume more energy in the *Initializing* and *Operating* states, and less energy in the *Idling* state.

Values can be specified using Action Language Helper (ALH) expressions supported by the [modeling tool](#) in:

- The body of the opaque behavior set as the *entry* or *do* behavior of the relevant state.

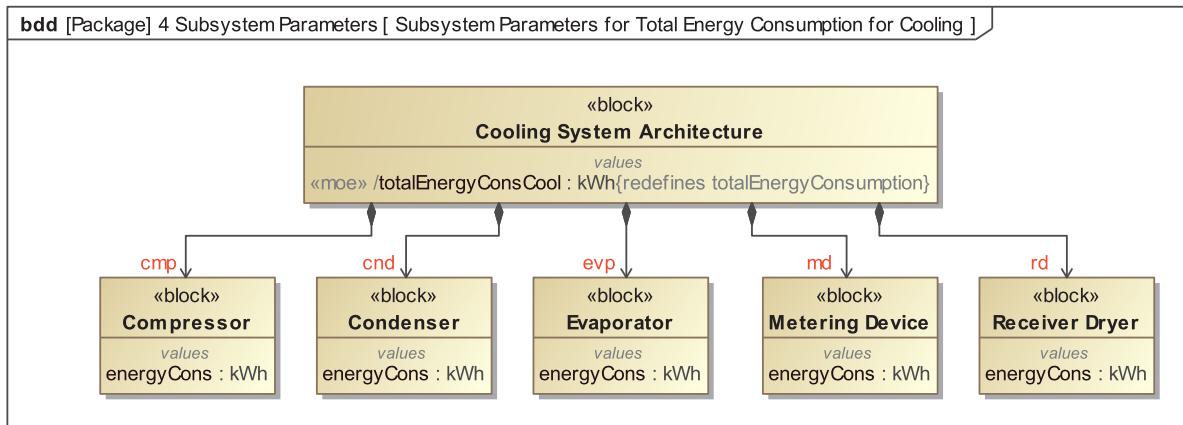


- The body of the opaque action which belongs to the *entry* or *do* behavior of the relevant state.

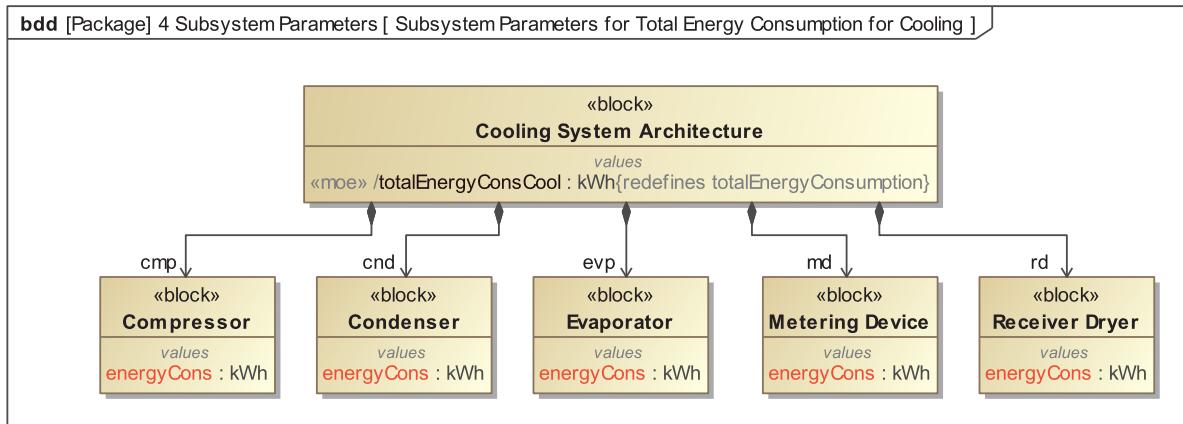


As you can see, this *ALH.setValue* expression has a set of variables that require:

1. The name of the part property (highlighted in the following figure) that is typed by the block whose value property you want to update. That part property must belong to the block which encloses the state machine diagram; in this case, it is the *Cooling System Architecture* block.



2. The name of the value property you want to update (highlighted in the following figure), in quotes. The value property must belong to the block which types the previously defined part property.

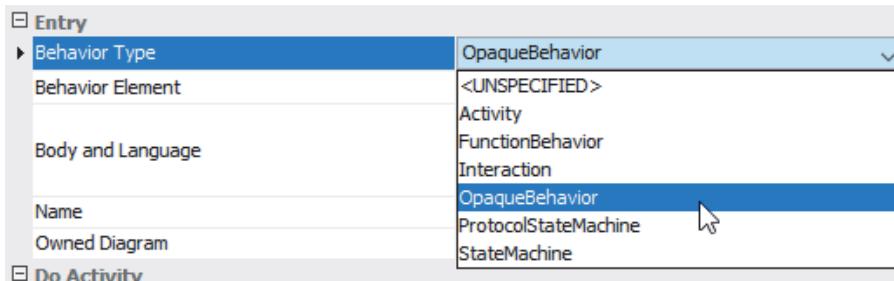


3. Concrete value you want to set for the previously defined value property (for example, 0.5).

To specify values in the body of the opaque behavior set as the *entry* behavior of the *Idling* state of the Cooling System

1. Open the *Cooling System States* diagram:
 - a. Press Ctr + Alt + F.
 - b. Select **Diagram** and type *cooling system s*.
 - c. When you see the *Cooling System States* diagram in the search results below, press Enter.
2. Double-click the shape of the *Idling* state.
3. On the right side of the open dialog, under the **Entry** category, select the **Behavior Type** property.

4. Click the value cell of this property and from the drop-down list, select **Opaque Behavior**.

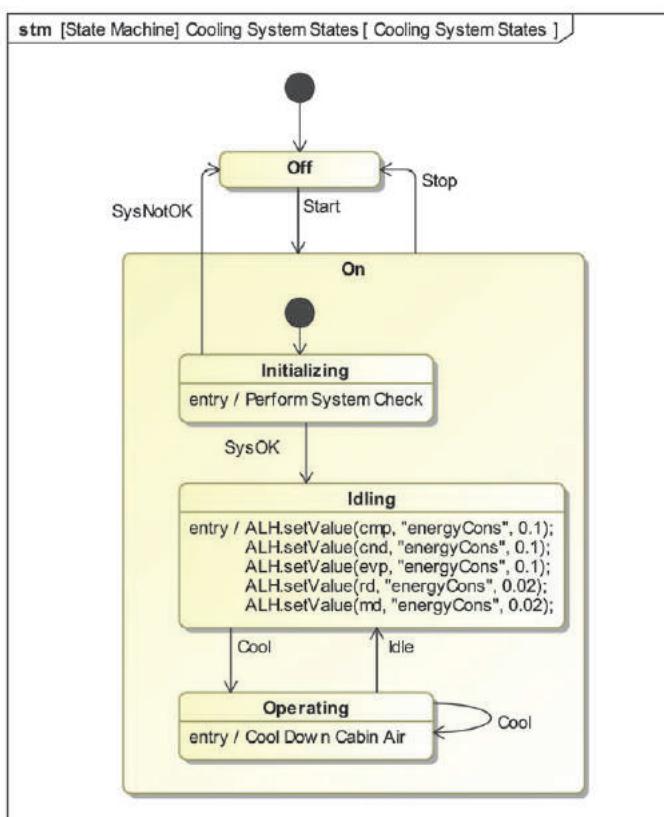


5. Select the nearest **Body and Language** property below.

6. Click the value cell of this property and type there:

```
ALH.setValue(cmp, "energyCons", 0.1);
ALH.setValue(cnd, "energyCons", 0.1);
ALH.setValue(epv, "energyCons", 0.1);
ALH.setValue(rd, "energyCons", 0.02);
ALH.setValue(md, "energyCons", 0.02);
```

7. Click **Close**. The shape of the *Idling* state displays the ALH expressions.



To specify the values in the body of the opaque action which belongs to the *entry* behavior of the *Initializing* state

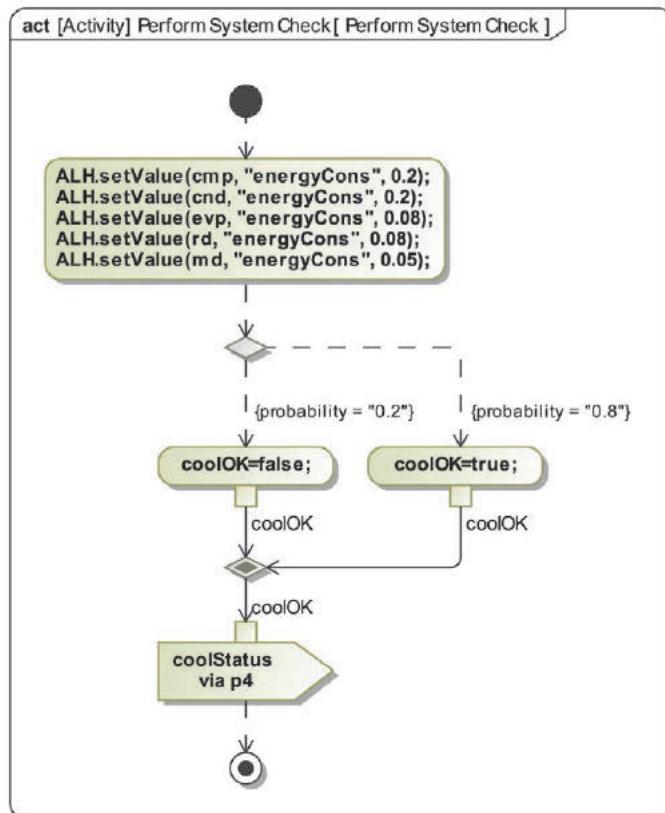
1. Open the *Perform System Check* diagram:

- Press Ctr + Alt + F.
- Select **Diagram** and type *perform*.
- When you see the *Perform System Check* diagram in the search results below, press Enter.

2. Click the small black triangle next to the **Action** button on the diagram palette and then select **Opaque Action**.
3. Click the control flow between the initial and decision nodes. A new opaque action is created and displayed on the diagram.
4. Immediately type on the shape of the newly created opaque action:

```
ALH.setValue(cmp, "energyCons", 0.2);
ALH.setValue(cnd, "energyCons", 0.2);
ALH.setValue(epv, "energyCons", 0.08);
ALH.setValue(rd, "energyCons", 0.08);
ALH.setValue(md, "energyCons", 0.05);
```

5. Press Enter.



Following the procedure above, you can update the *Cool Down Cabin Air* diagram. When finished, you can run the simulation to see how the *totalEnergyConsCool* value property changes the value over various states of the Cooling System. Make sure you run the simulation with the context (the *Cooling System Design* block) which encloses the state machine diagram; otherwise, the subsystem parameter value is not calculated. For this, click the small black triangle next to the Run button and select **Run with Context**.

Subsystem parameters done. What's next?

Finishing with the **Subsystem Parameters** cell means that you're done with the entire model, which describes the solution architecture of the logical subsystem. The only task before you can integrate it into the model of the whole **Sol** is establishing traceability relationships from the elements that capture the LSSA to subsystem requirements.

Traceability to Subsystem Requirements

What is it?

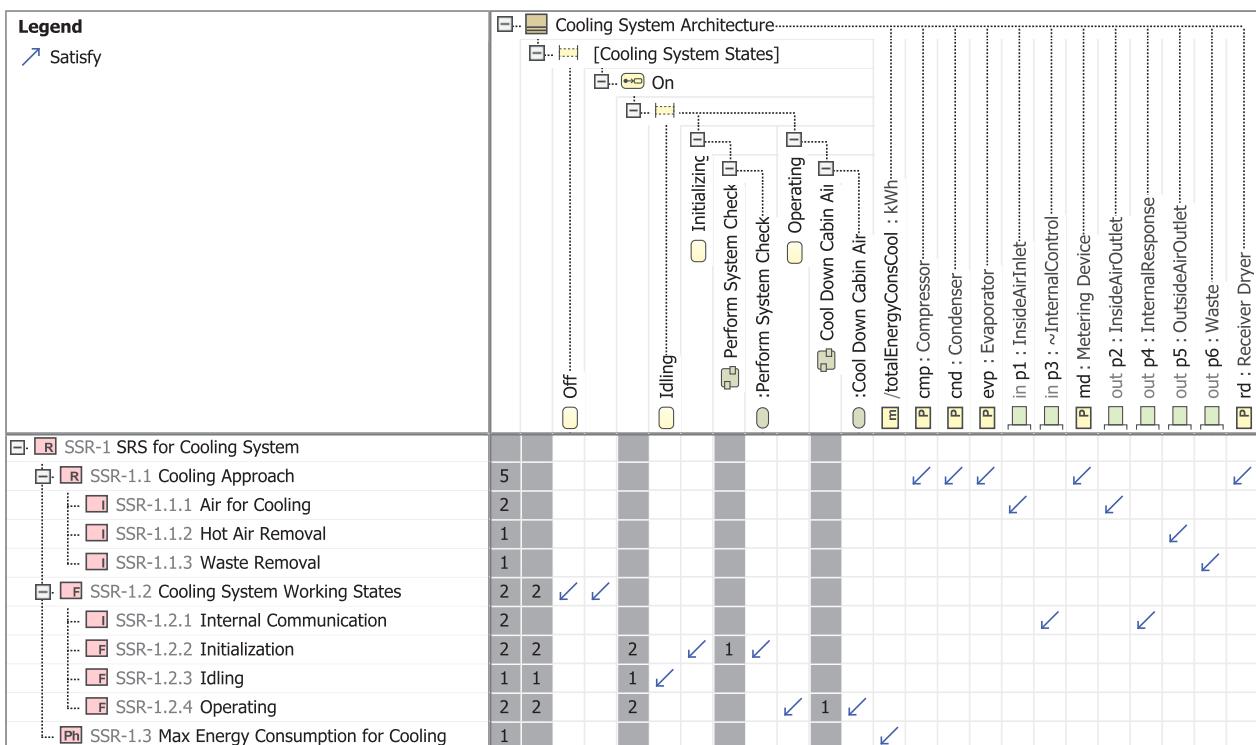
To have a complete LSSA model of a particular logical subsystem, you need to establish traceability relationships from the elements that capture the LSSA to the elements that capture the subsystem requirements. As determined by SysML, these are the satisfy relationships, meaning that one of the latter fulfills one of the former; for example, a satisfy relationship can be established from the part property typed by the block that captures some logical component to a certain subsystem requirement. In the complete LSSA model, all subsystem requirements are satisfied by one or more architecture elements even though these elements are not yet implemented.

Who is responsible?

Traceability relationships in the LSSA model of the appointed logical subsystem can be captured by the Systems Architect or Systems Engineer who belongs to the engineering team responsible for building the LSSA of that logical subsystem.

How to model?

Neither of the SysML diagrams is suitable for creating a mass of cross-cutting relationships, such as satisfy. For this, a dependency matrix 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.



Tutorial

- Step 1. Creating a dependency matrix for capturing satisfy relationships
- Step 2. Capturing satisfy relationships to subsystem requirements
- Step 3. Traceability to and revision of system requirements

Step 1. Creating a dependency matrix for capturing satisfy relationships

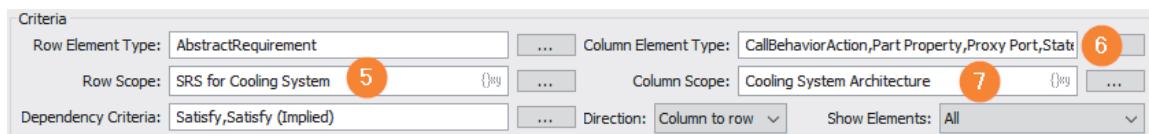
In this step, we create a Satisfy Requirement Matrix and specify its criteria. Subsystem requirements can be displayed in the rows of the matrix, and the elements that capture the LSSA can be displayed in its columns.

To create a Satisfy Requirement Matrix

1. In the Model Browser, right-click the *5 Traceability* package (you created it in [step 4](#) in the [Subsystem Requirements](#) tutorial) and select **Create Diagram**.
2. In the search box, type *srm* (the acronym of the predefined Satisfy Requirement Matrix), and press Enter.

(i) 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 *LSSA to Subsystem Requirements* 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 *SSR-1 SRS for Cooling System* requirement (you might need to expand the *1 Subsystem Requirements* package first) and drag it onto the **Row Scope** box in the **Criteria** area above the matrix contents. The *SSR-1 SRS for Cooling System* requirement becomes the scope of the matrix rows (see the following figure).
6. Select column element types:
 - a. In the Model Browser, select any part property, proxy port, value property, state, and call behavior action.

(i) 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.
 - b. Drag the selection onto the **Column Type** box in the **Criteria** area above the matrix contents. The columns of the matrix are set to display all the part properties, proxy ports, value properties, states, and call behavior actions (see the following figure) within the scope you select in the next step.
7. Select the column element scope:
 - a. In the Model Browser, select the *Cooling System Architecture* block (you might need to expand the *1 Subsystem Structure* package beforehand).
 - b. Drag the selection onto the **Column Scope** box in the **Criteria** area above the matrix contents. The selected block becomes the scope of the matrix columns (see the following figure).



Once you're done with the criteria, the contents of the matrix is updated. All the cells are empty at the moment, except the one at the intersection of the `totalEnergyConsCool` value property and the `SR-1.3 Max Energy Consumption for Cooling` requirement (as you may remember, this relationship was created in [step 6](#) of the [Subsystem Parameters](#) tutorial).

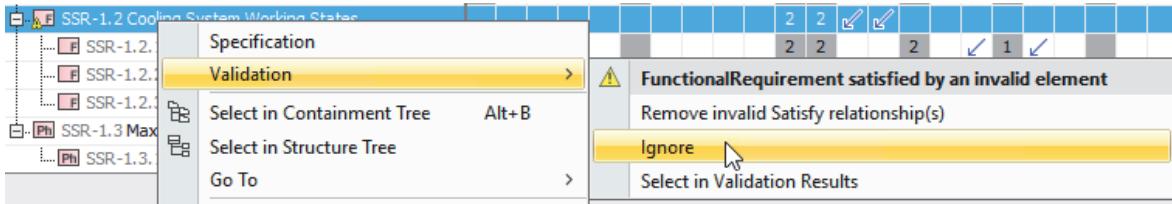
Legend	[Cooling System Architecture]					
	[Cooling System States]					
	On					
	Off					
SSR-1 SRS for Cooling System						
SSR-1.1 Cooling Approach						
SSR-1.1.1 Air for Cooling						
SSR-1.2 Cooling System Working States						
SSR-1.2.1 Internal Communication						
SSR-1.2.2 Initialization						
SSR-1.2.3 Idling						
SSR-1.2.4 Operating						
SSR-1.3 Max Energy Consumption for Cooling	1					

Step 2. Capturing satisfy relationships to subsystem requirements

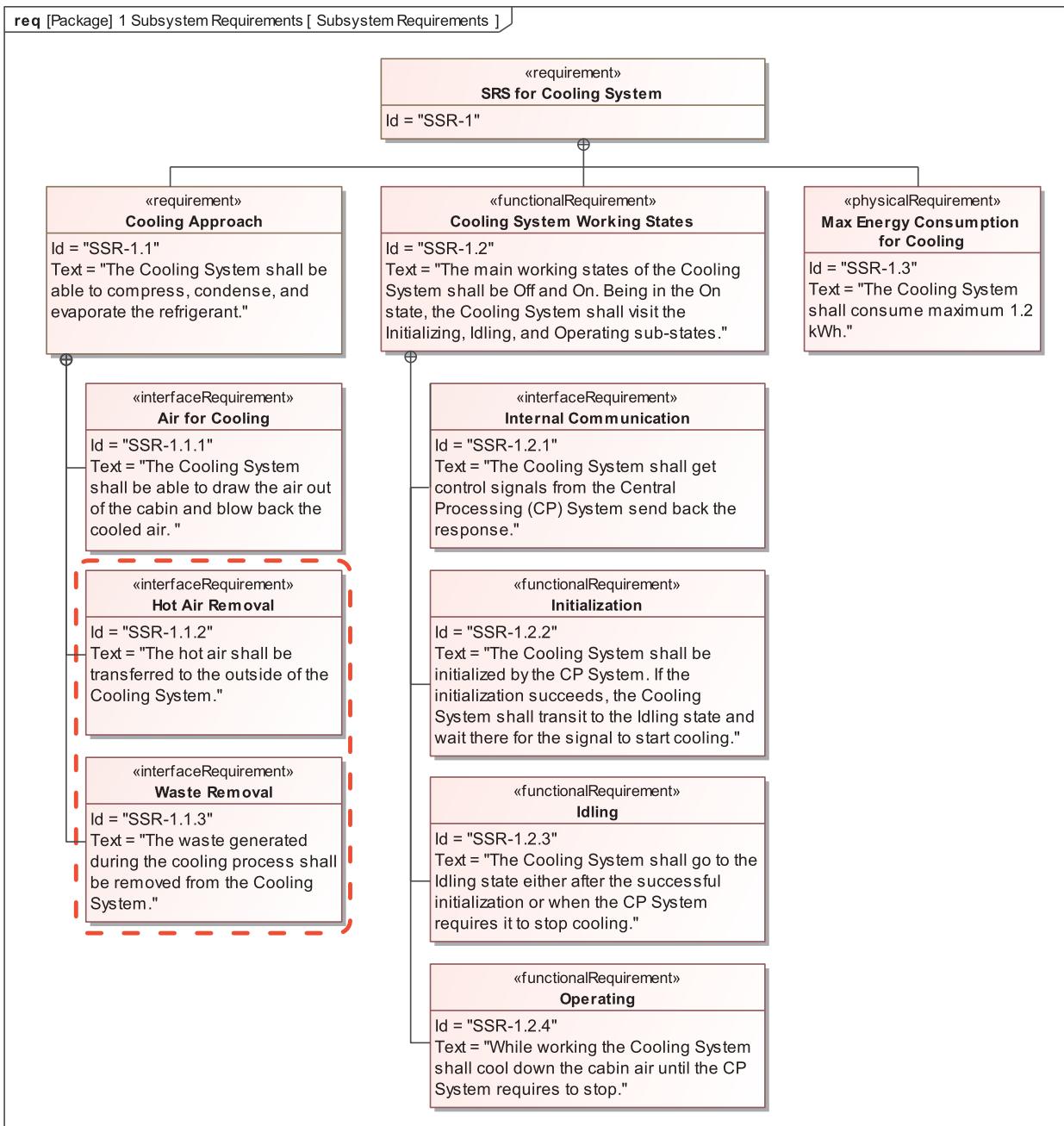
Now you're ready to establish other satisfy relationships. As you already know how to do this, we assume that you continue filling in the matrix yourself. If you need a hint as to what elements can be related, see the following figure.

Legend	[Cooling System Architecture]					
	[Cooling System States]					
	On					
	Off					
SSR-1 SRS for Cooling System						
SSR-1.1 Cooling Approach						
SSR-1.1.1 Air for Cooling	5					
SSR-1.2 Cooling System Working States	2					
SSR-1.2.1 Internal Communication	2					
SSR-1.2.2 Initialization	2					
SSR-1.2.3 Idling	2	2	2	2	2	2
SSR-1.2.4 Operating	1	1	1	1	1	1
SSR-1.3 Max Energy Consumption for Cooling	1					

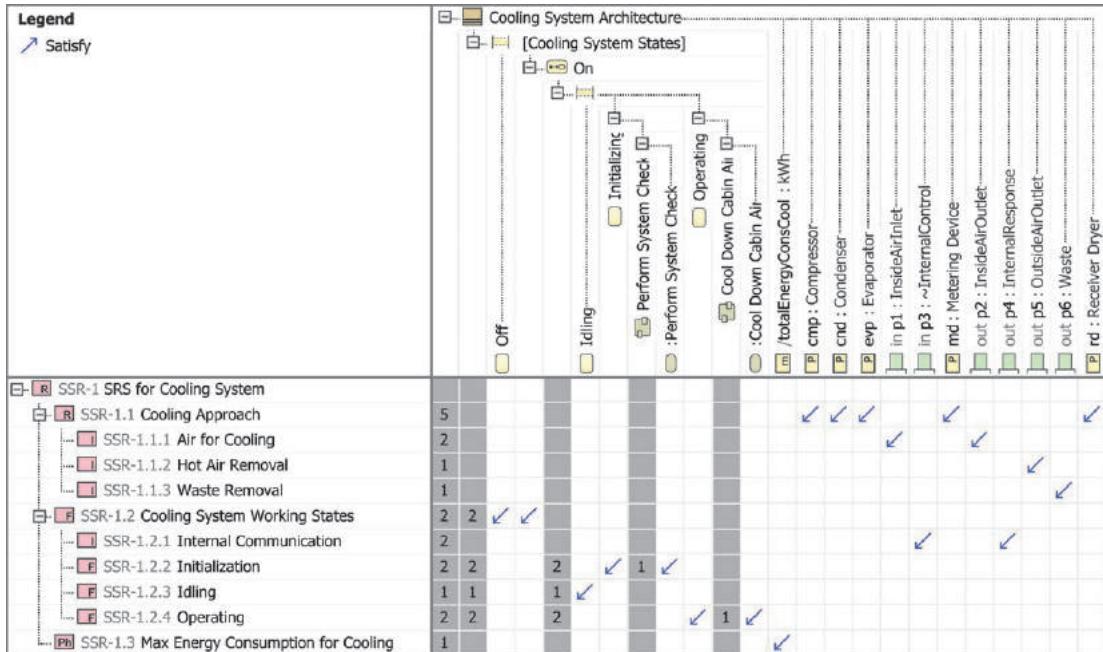
- i** If functional requirements become marked as invalid, there is no need to worry; you can simply ignore the warning.



As you can see, there are a few proxy ports that cannot trace to any subsystem requirement (as you can remember, they were identified in Chapter [Subsystem Structure](#)). This requires the revision and update of the subsystem requirements specification, and vice versa. This time, only the subsystem requirements specification needs to be updated. You can see the new requirements highlighted in the following figure.

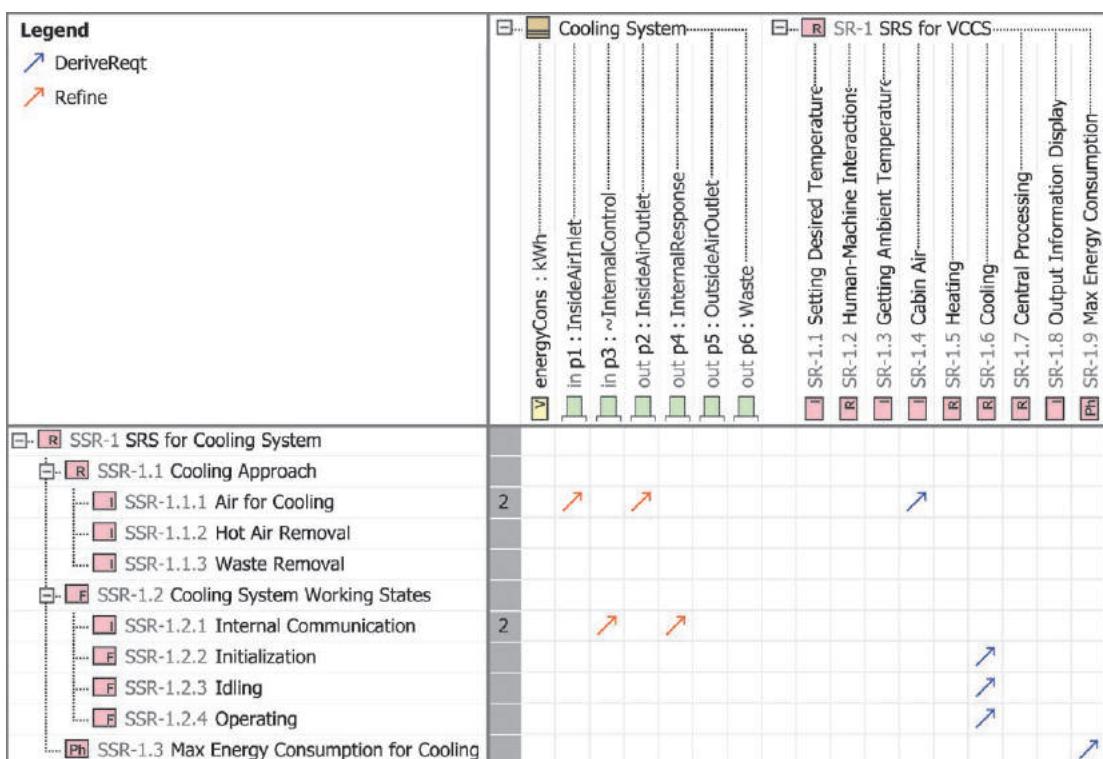


Afterwards, the satisfy requirements matrix in your model can be completely filled in. As you can see in the following figure, there are no hanging elements any more.

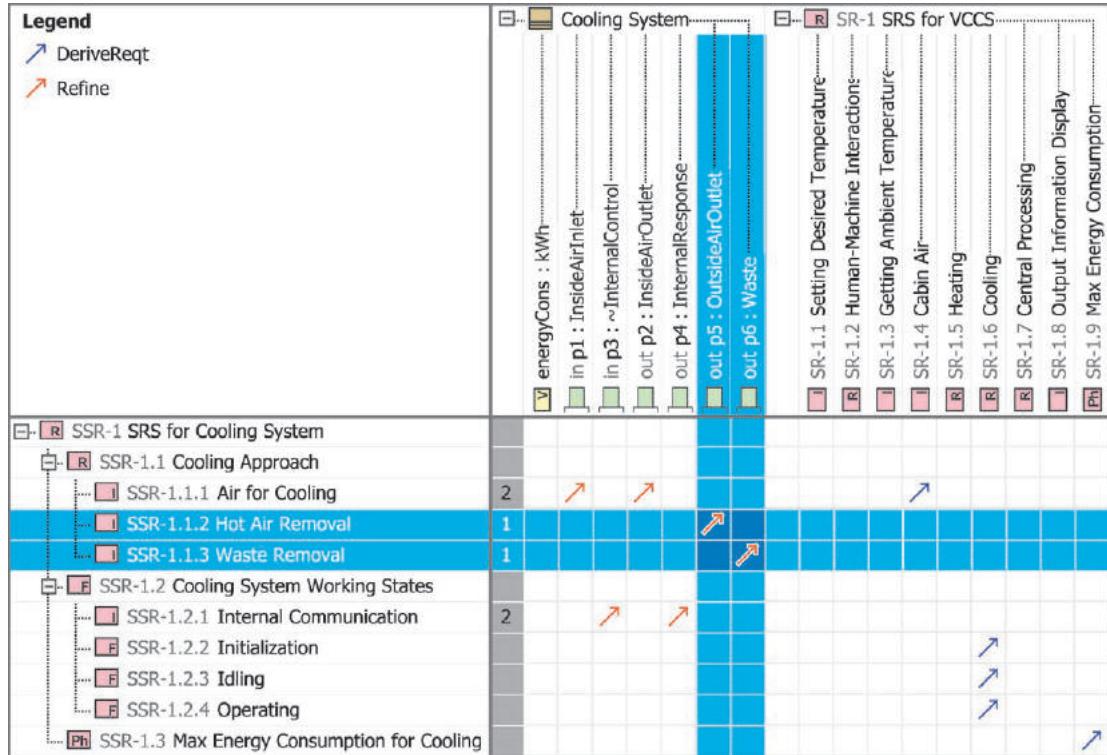


Step 3. Traceability to and revision of system requirements

If you look at the *Subsystem Requirements to System Requirements and LSA* matrix, you can see there are a few new subsystem requirements, which don't have traceability relationships. These are the subsystem requirements that you created in the previous step: *SSR-1.1.2 Hot Air Removal* and *SSR-1.1.3 Waste Removal*.



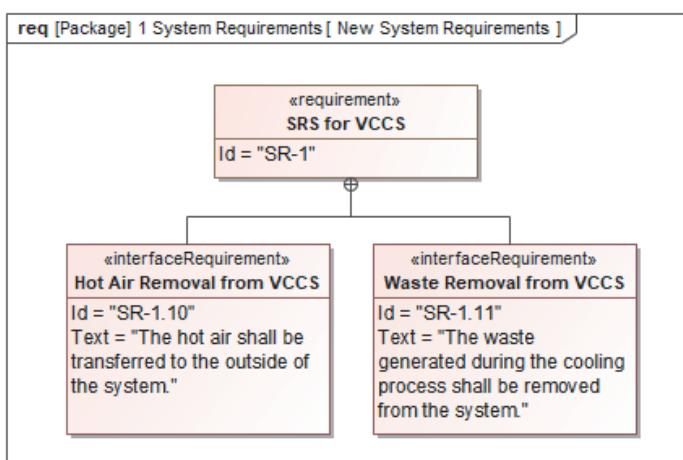
After you establish a few more refine relationships in this matrix, it should look like in the following figure.



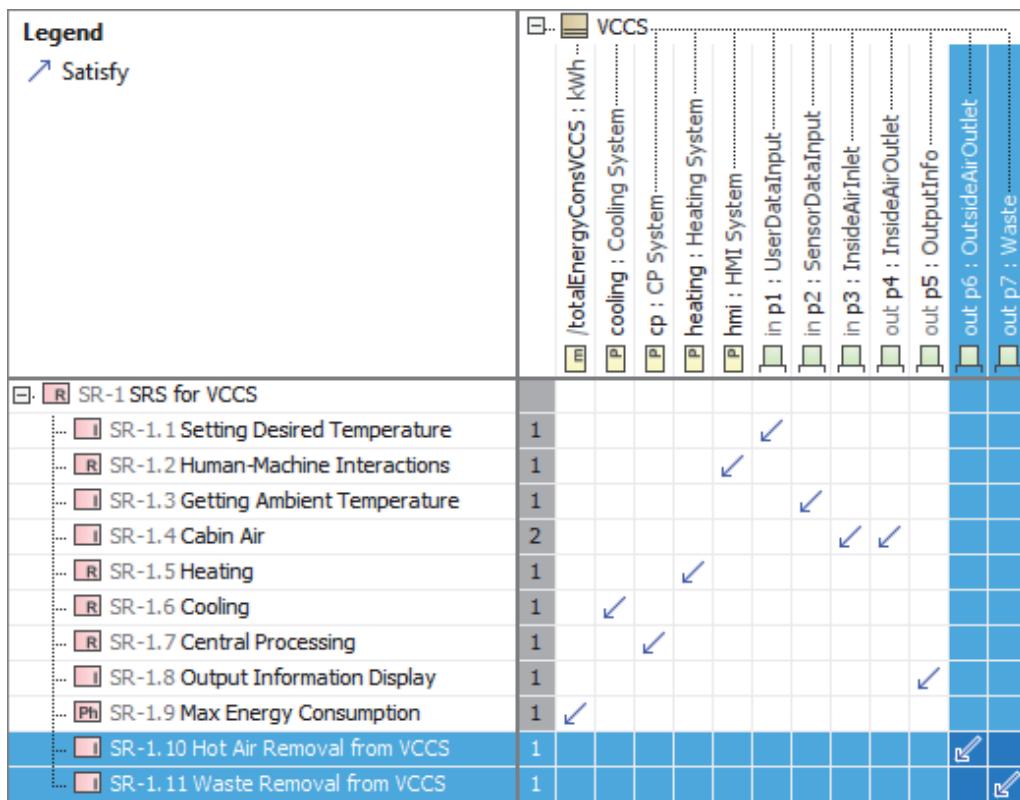
Now it's time to establish derivation relationships from these subsystem requirements to system requirements. This requires you to capture new system requirements first. The *SSR-1.1.2 Hot Air Removal* or *SSR-1.1.3 Waste Removal* requirements were created because of interfaces identified by the engineering team working on the LSSA model of the Cooling System. Moreover, these interfaces belong to the entire **Sol**, first of all. Therefore, we can state that at least two system requirements are currently missing.

Here is the action plan dealing with this situation:

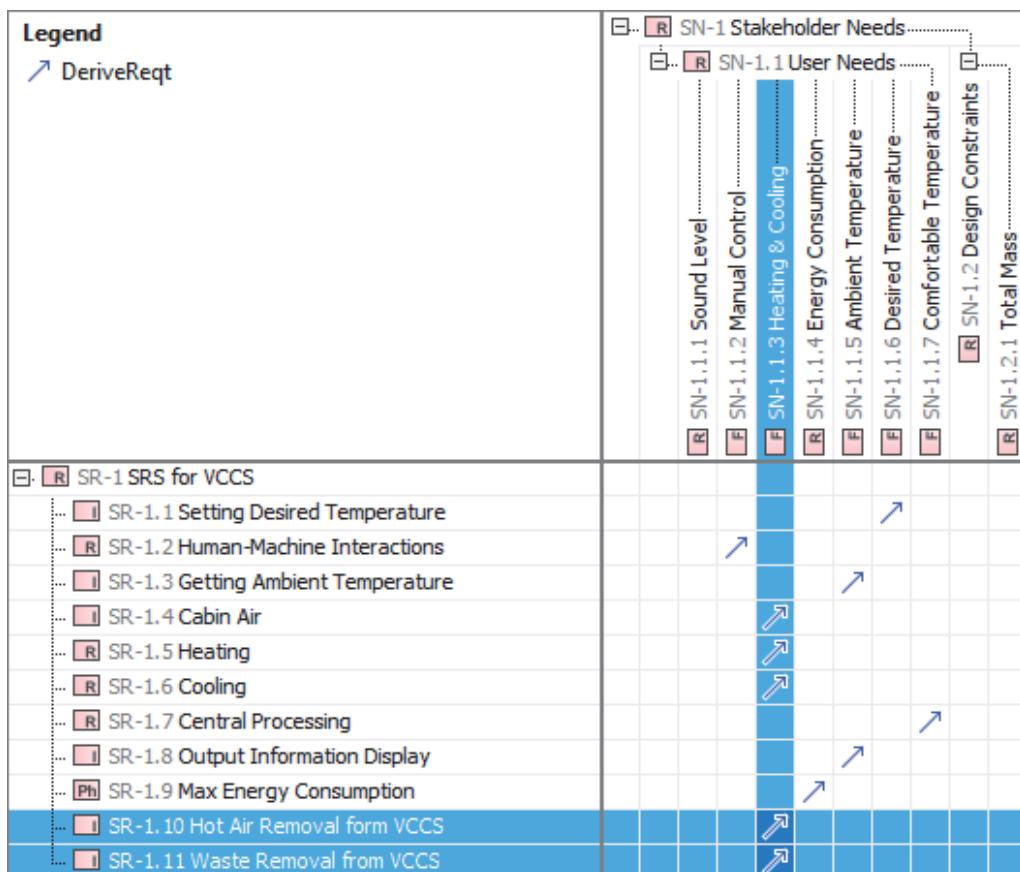
- As the systems architect, append the system requirements specification in the LSA model, with two new interface requirements.



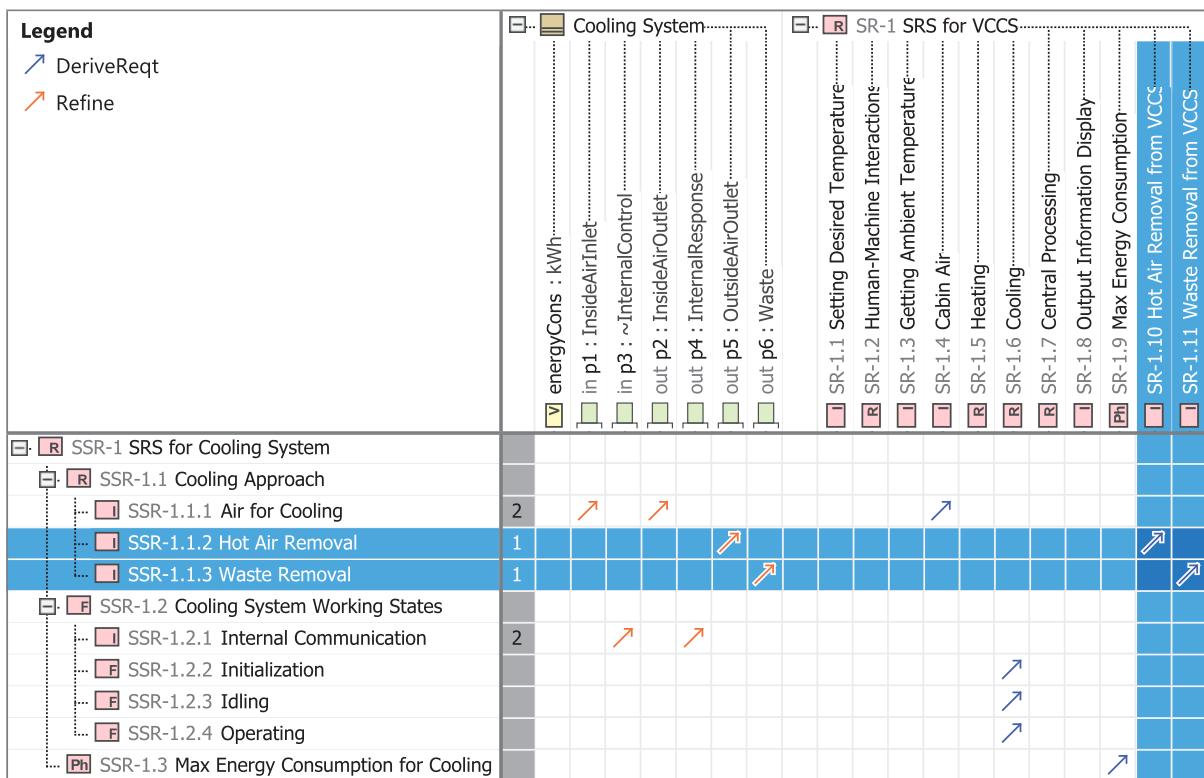
2. In the *LSA to System Requirements* matrix, establish satisfy relationships from relevant proxy ports (created in **step 5** of the **Subsystem Structure** tutorial) to these requirements.



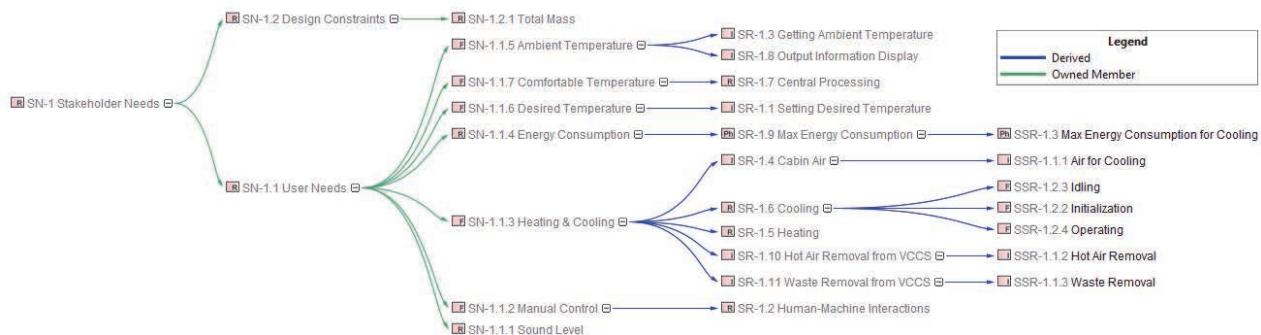
3. In the *System Requirements to Stakeholder Needs* matrix, establish missing derivation relationships from new system requirements to stakeholder needs.



4. As the appointed engineering team, establish missing derivation relationships between subsystem and new system requirements in the Cooling System SA model.



When you're done, look at the *From Stakeholder Needs to Subsystem Requirements* map. It should also be updated.



LSSA done. What's next?

The MagicGrid BoK does not detail how to model each component (i.e., Compressor, Condenser, and Evaporator, etc.) or even more precise units as defined by the MagicGrid framework. It's enough to note that the modeling workflow in these levels of detail is adequate to modeling in the subsystem level. That is, you start with derivation of requirements at that level of detail and then work on the structure, behavior, and parameters to satisfy them; once you've finished, you establish traceability relationships in order to approve the suggested logical architecture.

As mentioned in the [Preface](#), the component-level model of the solution domain can be both **MBSE** and **MBD**, because this is where the logical architecture relates to the logical (high-level) design of the **Sol**, and **MBD** starts. Therefore, besides SysML, other standards can be applied at the component-level. These

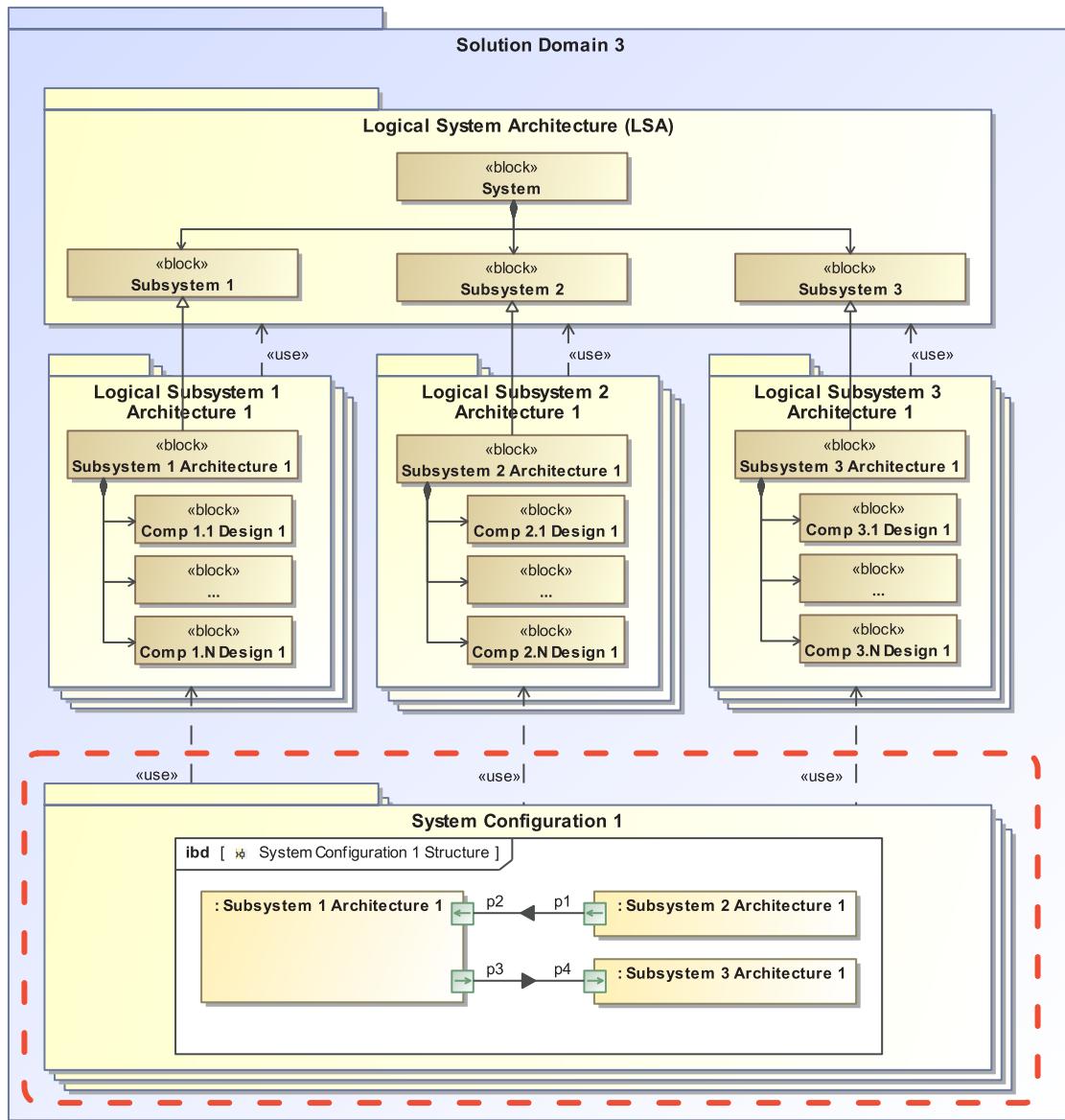
include OMG® Unified Modeling Language (UML)® for designing software components, Modelica® for designing mechanical, electrical, and electronic components, among others. Please be aware that for UML, you don't need to switch software, as UML is supported by the [modeling tool](#) along with SysML.

When you have the LSSA models of all logical subsystems, including their components and even more precise units, you can integrate them and have the entire logical architecture of the [Sol](#) in a single model (see Chapter [Building system configuration model](#)).

Building system configuration model

Building the system configuration model is the final phase of specifying the solution domain. It is performed after engineering teams are done with solution architectures of all logical subsystems of the **Sol**. The goal of this phase is to build the integrated model of the entire **Sol** and check if its logical subsystems can successfully interact with each other by verifying the integrity of the model. This is the task of the systems architect or systems integrator. Note that more than one solution can be proposed; the systems architect/integrator would perform a trade-off study to pick a preferred one of each logical subsystem.

The highlighted area in the following figure show the part of the solution domain we are talking about.



As you may conclude looking at the design layout of the MagicGrid framework (see the highlighted area in the following figure), the system configuration model captures the overall structure, behavior, and parameters of the **Sol**.

	Pillar						
			Requirements	Structure	Behavior	Parameters	Safety & Reliability
Domain	Problem	Black Box	Stakeholder Needs	System Context	Use Cases	Measures of Effectiveness (MoEs)	Conceptual and Functional Failure Mode & Effects Analysis (FMEA)
	White Box			Conceptual Subsystems	Functional Analysis	MoEs for Subsystems	Conceptual Subsystems FMEA
	Solution		System Requirements ^{Config}	System Structure ^{Config}	System Behavior ^{Config}	System Parameters ^{Config}	System Safety & Reliability (S&R)
			Subsystem Requirements	Subsystem Structure	Subsystem Behavior	Subsystem Parameters	Subsystem S&R
			Component Requirements	Component Structure	Component Behavior	Component Parameters	Component S&R
	Implementation		Implementation Requirements				

System Configuration Structure

What is it?

Once all engineering teams deliver the LSSA models of each logical subsystem of the **Sol** (defined in the LSA model), the systems architect/integrator is able to finish his/her job. That is, he/she can integrate all the models into the whole and build the system configuration model.

The main purpose of building the system configuration model is to verify whether logical subsystems of the **system of interest** can successfully communicate. The systems architect/integrator must check whether interfaces of separate logical subsystems are compatible. If one or more incompatible interfaces are detected, the systems architect/integrator requests the responsible engineering team to review and update their model appropriately. After the engineering teams complete the request, the systems architect/integrator can make another attempt to integrate the models.

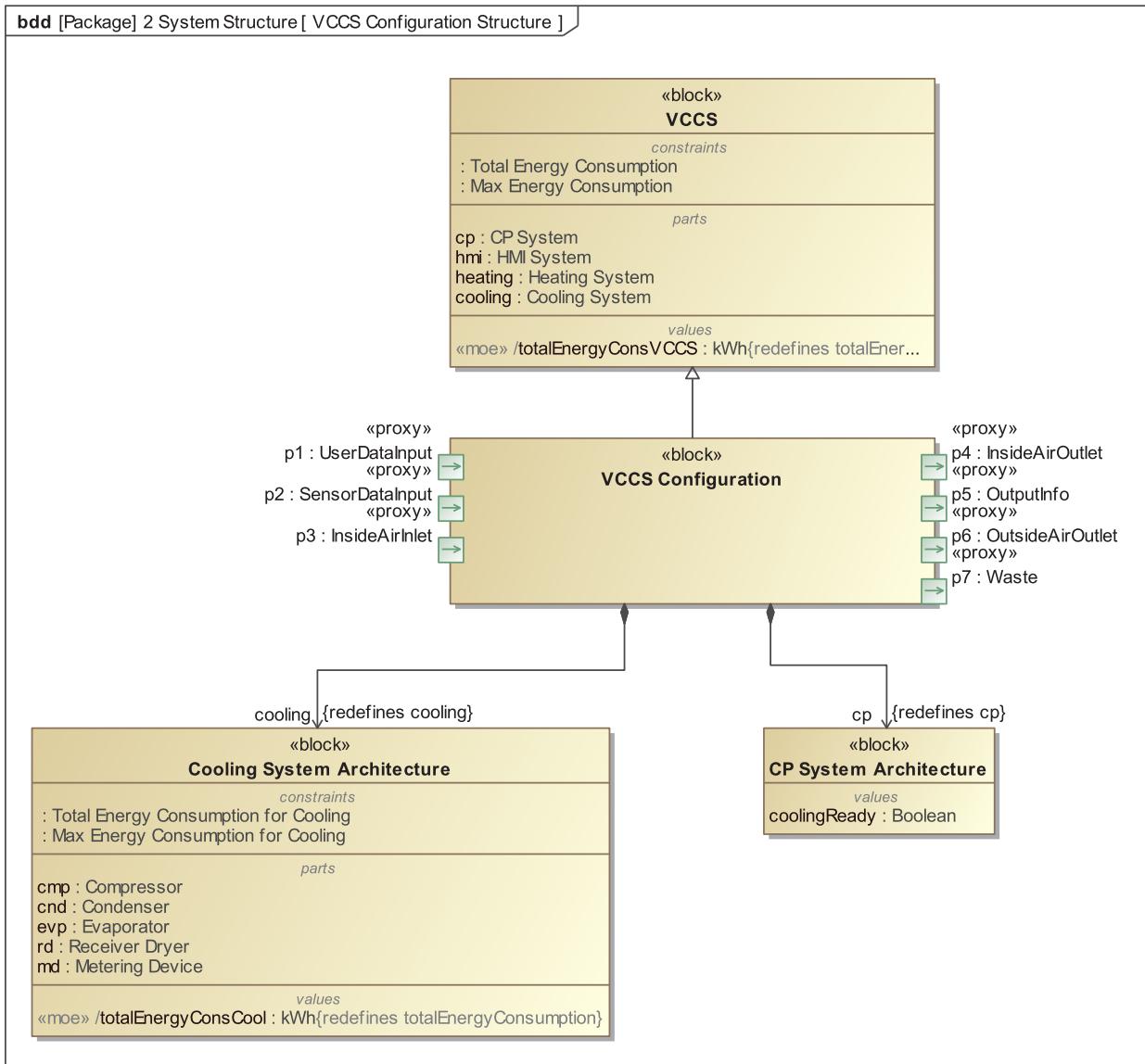
Although the VCCS has four logical subsystems overall, the following material focuses on integrating only two of them: the Cooling System and the CP System. The model with all logical subsystems integrated is available in <the installation folder of the **modeling tool**>\samples\MagicGrid; for example, C:\Program Files\Cameo Systems Modeler 2021\samples\MagicGrid.

Who is responsible?

The system configuration structure can be captured by the Systems Architect or Systems Engineer, who is the head of the whole systems engineering project and is responsible for the smooth integration of the subsystems and their components into the whole, the same person who owns the LSA model of the **Sol**. This task can also be performed by the Systems Integrator, a subordinate of the Systems Architect.

How to model?

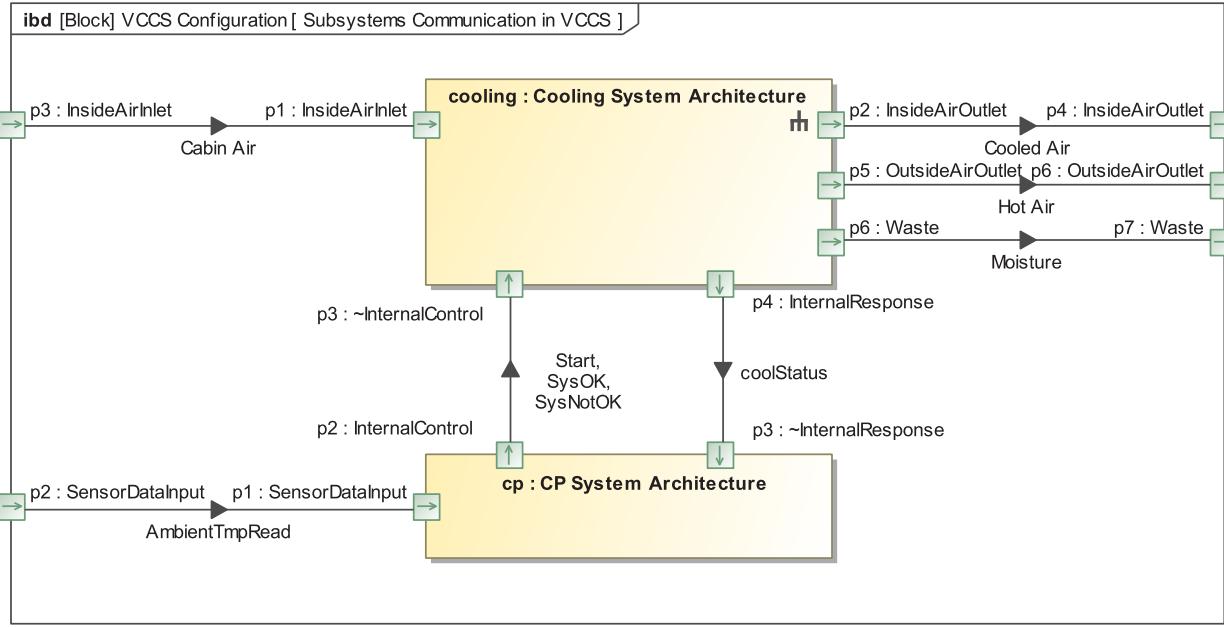
The integrated structure model of the system configuration can be captured by utilizing the infrastructure of the SysML bdd.



The structure decomposition map enables users to review and update the integrated structure model in a compact form.



The interface compatibility can be verified by specifying the logical subsystems communication in the SysML ibd.



Tutorial

- Step 1. Creating and organizing a model for system configuration structure
- Step 2. Getting ready to capture the structure of the system configuration
- Step 3. Capturing the integrated structure of the system configuration
- Step 4. Verifying interface compatibility

Step 1. Creating and organizing a model for system configuration structure

Since you already have the LSSA models of the Cooling System and CP System, you (in the role of the systems architect/integrator) can integrate them into a single model that represents the whole VCCS.

For this, you must create another model. 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 will focus on a single one.

To create a system configuration model of the VCCS

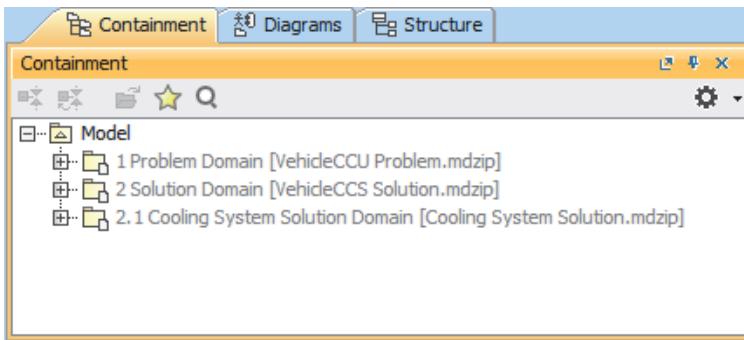
1. Start sharing the LSSA models of the Cooling System and the CP System.

- i**

 - The LSSA 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.

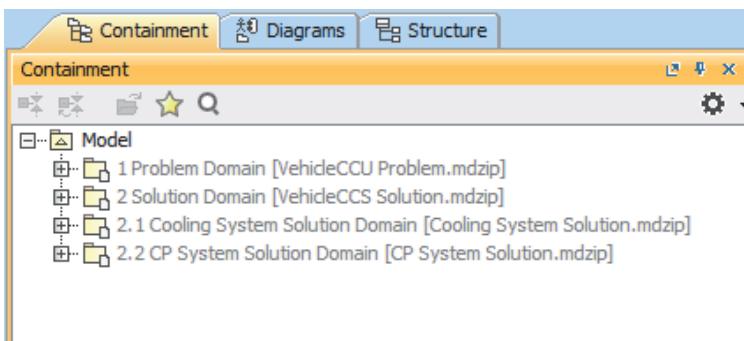
2. Create a new model. It can be named *VehicleCCS Configuration.mdzip*. This is the configuration model of the Vehicle Climate Control System.
3. Use the LSSA model of the Cooling System as *read-only* in the configuration model of the VCCS. Because it has already been used in the LSSA model of the Cooling System, the problem domain

model and the solution domain model of the VCCS (LSA model) are automatically used as well.

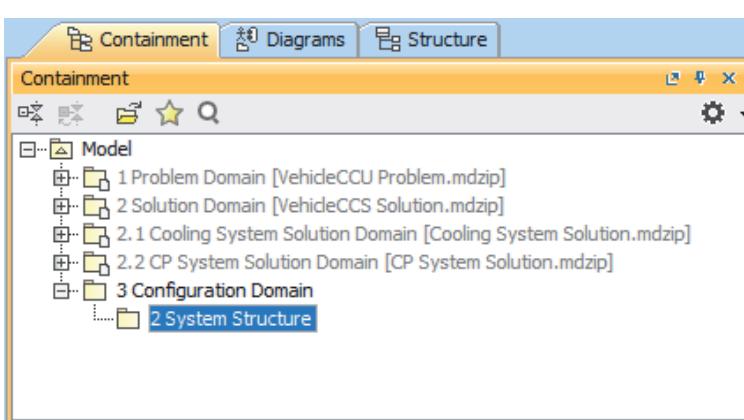


i Grey color element names within the used model indicate that elements cannot be modified. This is because the external model is used in the *read-only* mode.

4. Repeat step 3 to use the solution domain of the CP System.



5. Right-click the *Model* package (this is the default name of the root package) and select **Create Element**.
6. In the search box, type *pa* (the first two letters of the element type *Package*), and press Enter.
7. Type *3 Configuration Domain* to specify the name of the new package and press Enter (see the figure in step 9).
8. Right-click the *3 Configuration Domain* package and select **Create Element**.
9. Repeat step 6.
10. Type *2 System Structure* to specify the name of the new package and press Enter.



Step 2. Getting ready to capture the structure of the system configuration

To get ready to capture the integrated structure of the whole system of interest or one of its configurations, you first need to create a block which represents the system configuration. The block that captures the VCCS configuration must inherit all the structural and behavioral features from the block that captures the VCCS in the LSA model (see [step 3](#) of the [System Structure](#) tutorial). Thus, the block that

captures the VCCS configuration must be a sub-type of the the block that captures the VCCS in the LSA model.

A bdd created under the *2 System Structure* package can be used for capturing this block, as well as for performing the next steps of this cell tutorial.

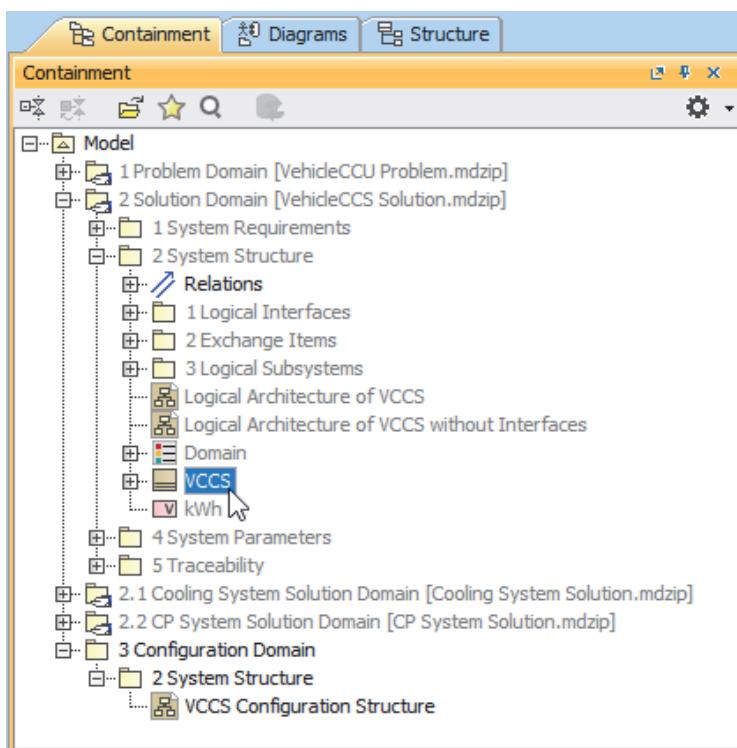
To get ready to capture the structure of the VCCS configuration

1. Create a bdd:

- a. In the Model Browser, right-click the *2 System Structure* package you created in [step 1 of this cell tutorial](#), 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 *VCCS Configuration Structure* to specify the name of the new diagram and press Enter again.

2. Display the *VCCS* block from the LSA model on the diagram:

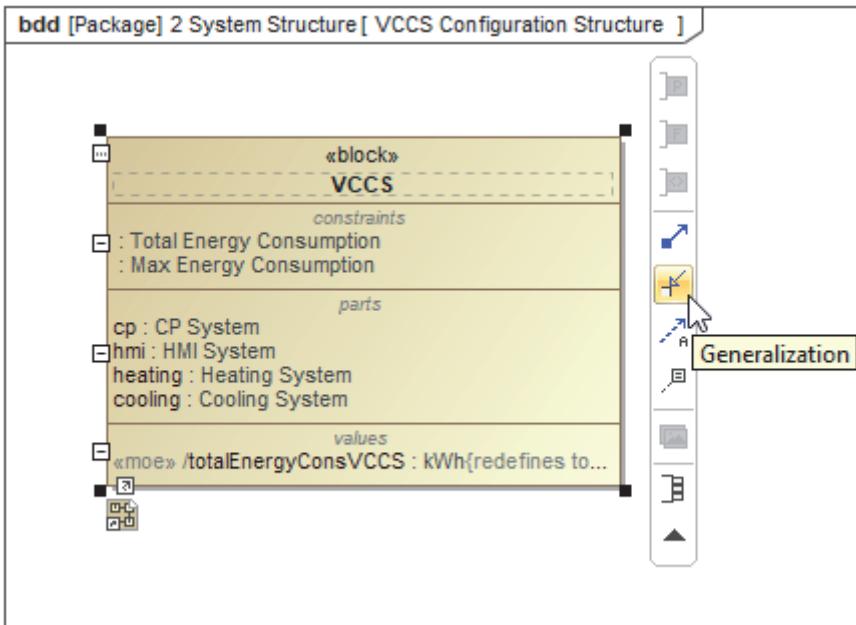
- a. In the Model Browser, select the *VCCS* block (you might need to expand the *2 Solution Domain* and *2 System Structure* packages first).



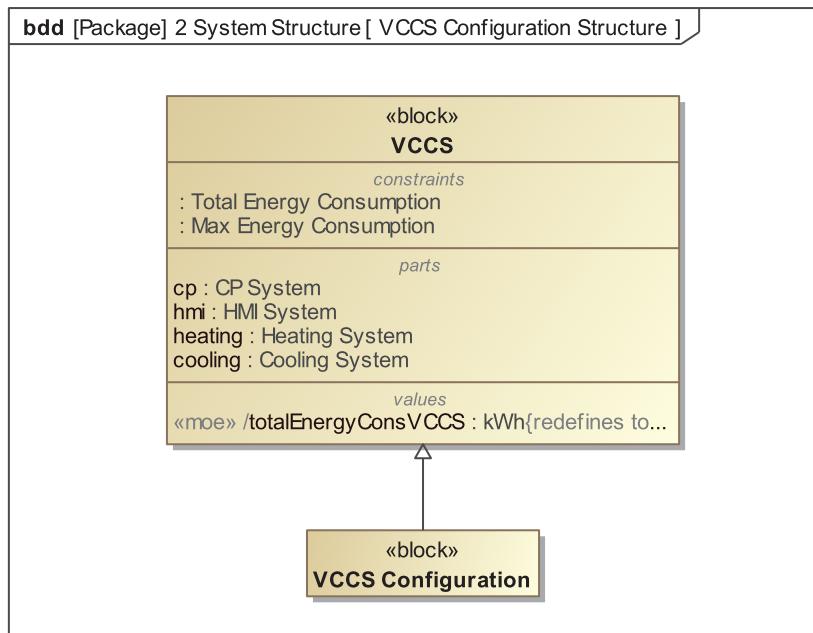
- b. Drag the selection to the newly created diagram pane. The shape of the *VCCS* block is displayed on the diagram.

3. Create a block to capture the VCCS configuration:

- Select the *VCCS* block, if not yet selected, and click the Generalization button  on its smart manipulator toolbar.



- Click an empty place on the diagram below. A new unnamed block is created, and its shape is displayed on the diagram pane.
- Type *VCCS Configuration* to specify the name of the new block and press Enter.



4. Redefine the inherited proxy ports:

- Double-click the shape of the *VCCS Configuration* block on the diagram.
- On the left of the open dialog, click **Ports/Interfaces**.

- c. Select the proxy port in the first row and click the **Redefine** button below.

Ports/Interfaces				
	Direction	Port Name	Port Type	Type Features
Proxy Port				
^		p1	UserDataInput [2 Solution Dom...]	[F] in dts : DesiredTmpSet
^		p2	SensorDataInput [2 Solution Do...]	[F] in atr : AmbientTmpRead
^		p3	InsideAirInlet [2 Solution Domain...]	[F] in ca : Cabin Air
^		p4	InsideAirOutlet [2 Solution Doma...]	[F] out ca : Comfortable Air
^		p5	OutputInfo [2 Solution Domain:...]	[F] out ss : vccsStatus
^		p6	OutsideAirOutlet [2 Solution Do...]	[F] out ha : Hot Air
^		p7	Waste [2 Solution Domain::2 Sy...]	[F] out m : Moisture

As a result, the selected proxy port is redefined. From there, it is no longer decorated with the caret symbol ("^"), which indicates the inheritance. Don't change anything.

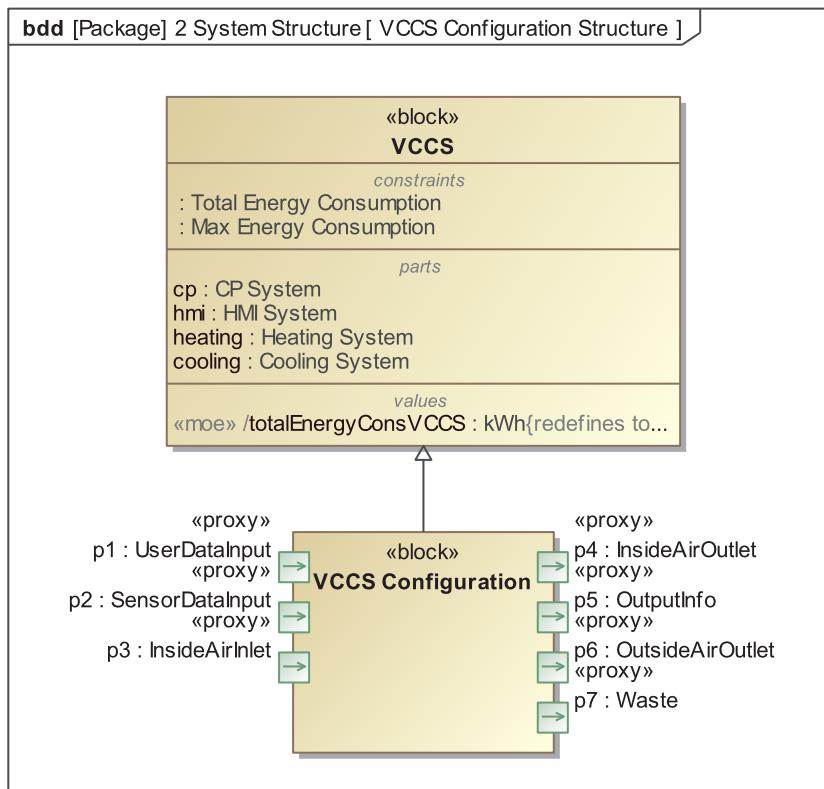
Ports/Interfaces				
	Direction	Port Name	Port Type	Type Features
Proxy Port				
^		p2	SensorDataInput [2 Solution Do...]	[F] in atr : AmbientTmpRead
^		p3	InsideAirInlet [2 Solution Domain...]	[F] in ca : Cabin Air
^		p4	InsideAirOutlet [2 Solution Doma...]	[F] out ca : Comfortable Air
^		p5	OutputInfo [2 Solution Domain:...]	[F] out ss : vccsStatus
^		p6	OutsideAirOutlet [2 Solution Do...]	[F] out ha : Hot Air
^		p7	Waste [2 Solution Domain::2 Sy...]	[F] out m : Moisture
		p1	UserDataInput [2 Solution Dom...]	[F] in dts : DesiredTmpSet

- d. Row by row, redefine the rest (the order of items may differ from what you see in the following figure).

Ports/Interfaces				
	Direction	Port Name	Port Type	Type Features
Proxy Port				
		p1	UserDataInput [2 Solution Dom...]	[F] in dts : DesiredTmpSet
		p2	SensorDataInput [2 Solution D...]	[F] in atr : AmbientTmpRead
		p3	InsideAirInlet [2 Solution Domai...]	[F] in ca : Cabin Air
		p4	InsideAirOutlet [2 Solution Doma...]	[F] out ca : Comfortable Air
		p5	OutputInfo [2 Solution Domain:...]	[F] out ss : vccsStatus
		p6	OutsideAirOutlet [2 Solution Do...]	[F] out ha : Hot Air
		p7	Waste [2 Solution Domain::2 S...]	[F] out m : Moisture

- e. Click **Close**.

5. Click the Display All Ports button  on the smart manipulator toolbar of the *VCCS Configuration* block. The shapes of proxy ports are displayed on the *VCCS Configuration* block.

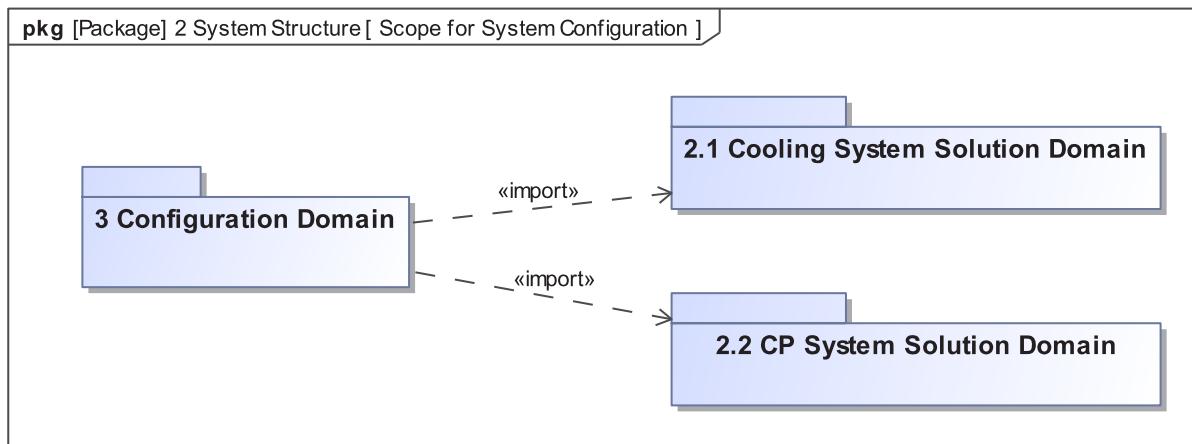


Step 3. Capturing the integrated structure of the system configuration

You can utilize the bdd created in the previous step to build the integrated structure model. All you need to do in this step is define the logical subsystems that are included in the particular system configuration. If (as in this case) every logical subsystem has only one solution architecture, the system configuration is also singular.

When specifying the system configuration structure, the blocks which represent the logical subsystems must be taken from the LSSA models; that is, the *Cooling System Architecture* block from the LSSA model of the Cooling System (the *Cooling System Solution.mdzip* file), and the *CP System Architecture* block from the LSSA of the CP System (the *CP System Solution.mdzip* file). Be sure you haven't taken the blocks representing the logical subsystems from the LSA model. These blocks have neither internal structure no behavior; remember, they were created to identify the work packages.

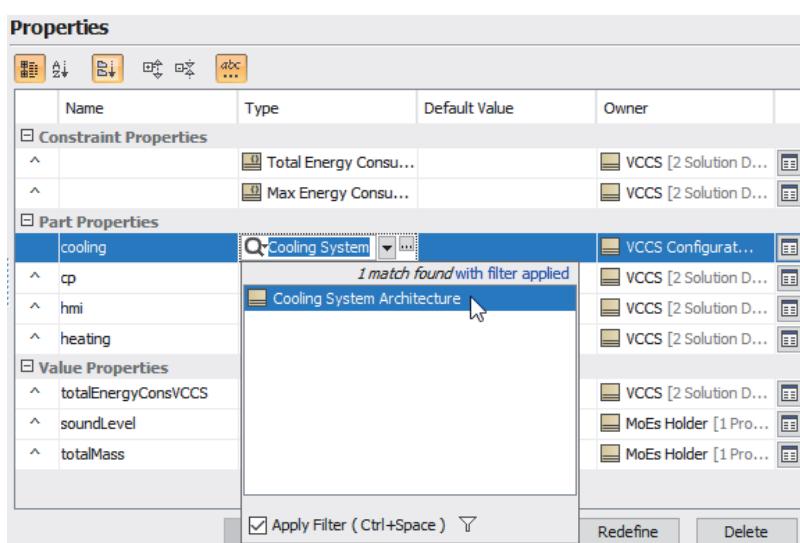
The **modeling tool** actually enables you to prevent the selection of inappropriate blocks. For this, you need to narrow the scope for type selection down to one or more packages. In this case, these are the *2.1 Cooling System Solution Domain* and *2.2 CP System Solution Domain* packages from the used models, as shown in the following figure. Since you are aware of how to narrow down the scope (see [step 3](#) of *System Structure* tutorial, if you need to refresh), we assume that you do this on your own.



Now you're ready to capture the integrated structure of the VCCS configuration. Just don't be too quick to create new part properties for the *VCCS Configuration* block. As a sub-type of the *VCCS* block, it already has them, inherited from the super-type. Therefore, all you need to do is redefine the inherited part properties of the *VCCS Configuration* block and select new types for them.

To capture the integrated structure of the VCCS configuration

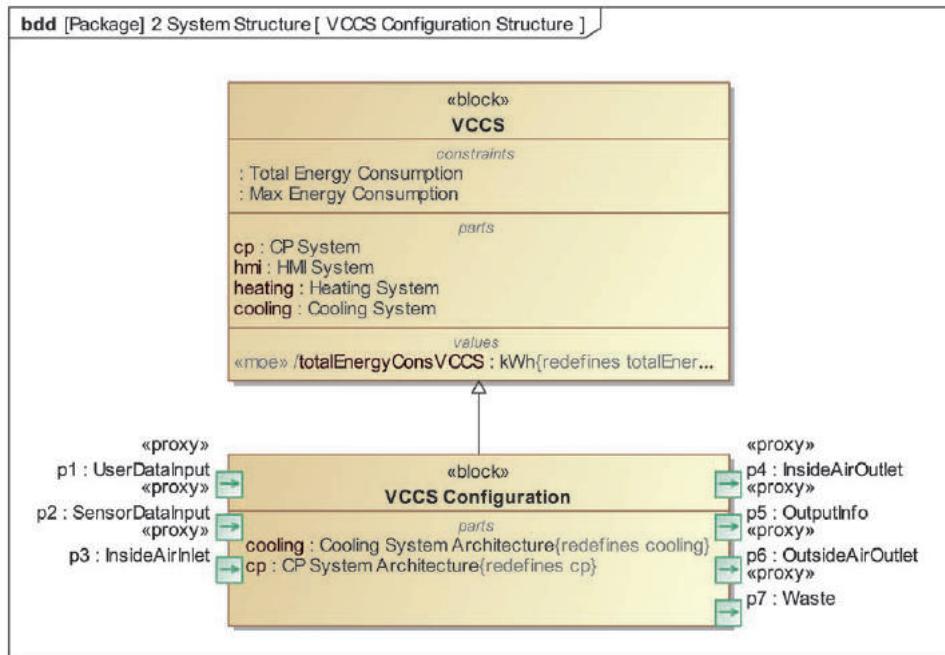
1. Open the *VCCS Configuration Structure* bdd, if not opened yet.
2. Integrate the LSSA of the Cooling System into the VCCS configuration:
 - a. Double-click the *VCCS Configuration* block.
 - b. In the Specification of the *VCCS Configuration* block, click **Properties** on the left.
 - c. Select the *cooling* part property on the right and then click the **Redefine** button below.
 - d. In the **Type** column, select the *Cooling System Architecture* block.



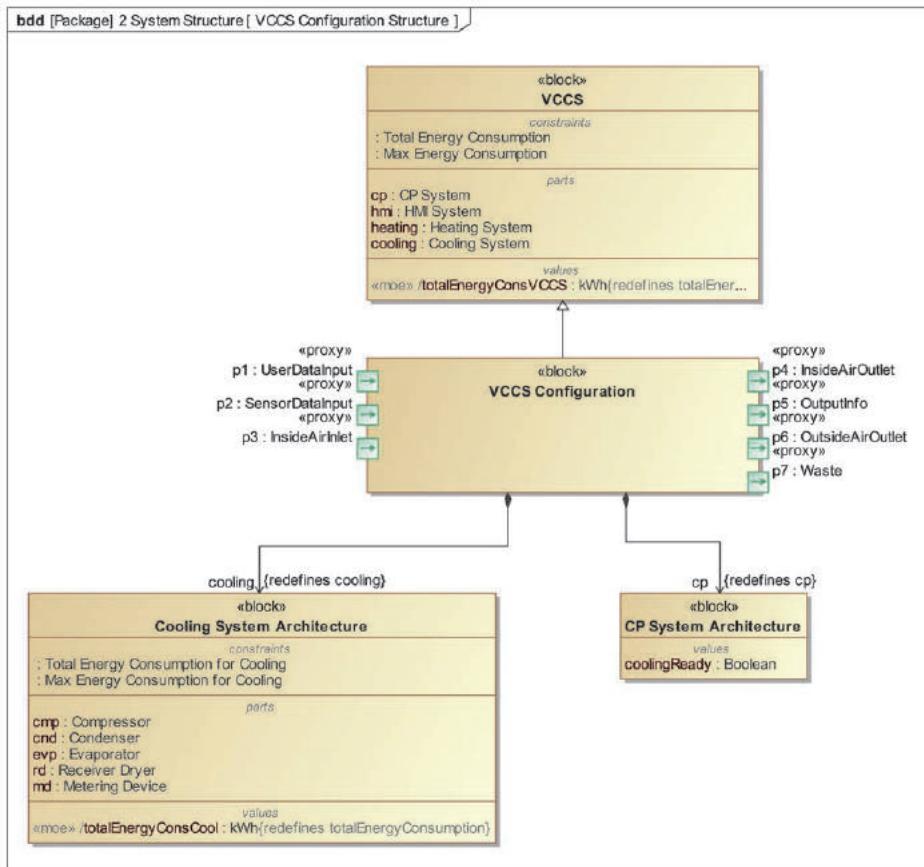
3. Follow steps 2.c and 2.d to integrate the LSSA of the CP System into the VCCS configuration.

Properties				
	Name	Type	Default Value	Owner
Constraint Properties				
^	Total Energy Consumption	[2 Solution Domains]	VCCS [2 Solutions]	
^	Max Energy Consumption	[2 Solution Domains]	VCCS [2 Solutions]	
Part Properties				
cooling	Cooling System Architecture	[2.1.1]	VCCS Configuration	
cp	CP System Architecture	[2.2 CP ...]	VCCS Configuration	
^ hmi	HMI System	[2 Solution Domains]	VCCS [2 Solutions]	
^ heating	Heating System	[2 Solution Domains]	VCCS [2 Solutions]	
Value Properties				
^ totalEnergyConsVCCS	kWh	[2 Solution Domain::2 Systems]	VCCS [2 Solutions]	
^ soundLevel	dBA	[1 Problem Domain::1 Black ...]	MoEs Holder	
^ totalMass	mass[kilogram]	[ISO-80000:ISO...	MoEs Holder	

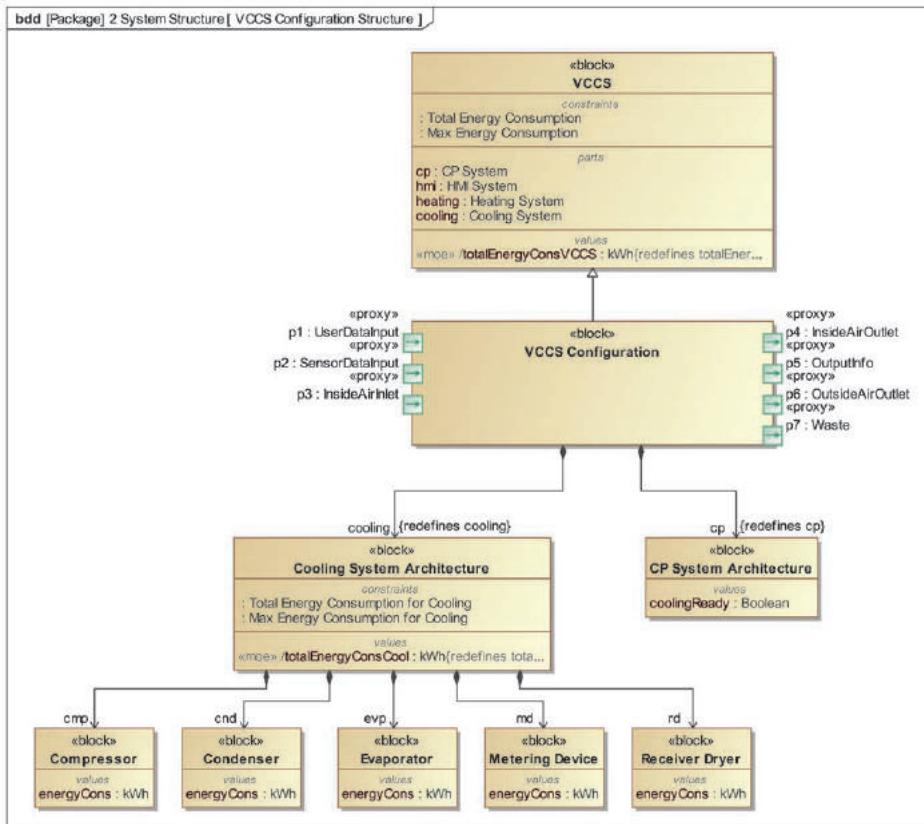
After you close the dialog, the *VCCS Configuration Structure* diagram should look very similar to the one below.



If you drag the parts, one by one, out of the enclosing block shape, the diagram should look like this.



If you decide to display the deeper structure (right-click the *Cooling System Design* block and select **Display** > **Display Related Elements**), the diagram should look as follows.



The integrated structure of the **Sol** configuration can also be displayed in the structure decomposition map, one of the predefined relation maps of the **modeling tool**. It can display any structure in a compact form, which is especially suitable for review.

To display the integrated structure of the VCCS configuration in a map

1. In the open diagram, select the shape of the *VCCS Configuration* block.
 2. Right-click the selection and then select **Create Diagram**.
 3. In the search box, type *sdm* (the acronym of the structure decomposition map), and then press Enter. The map is created.
- i** If you don't see any results, click the **Expert** button below the search results list. The list of available diagrams expands in the Expert mode.
4. Type *VCCS Configuration Structure* to specify the name of the new map and press Enter again.



- i** If your **modeling tool** version is lower than 2021x, you should see the *cooling* and *cp* part properties twice (see the following figure). This is because the structure decomposition map in these versions displays the inherited part properties even if they are redefined.



To remove the inherited part properties from the map, select them and press Delete.

Step 4. Verifying interface compatibility

Now it's time to specify how logical subsystems communicate within the VCCS configuration.

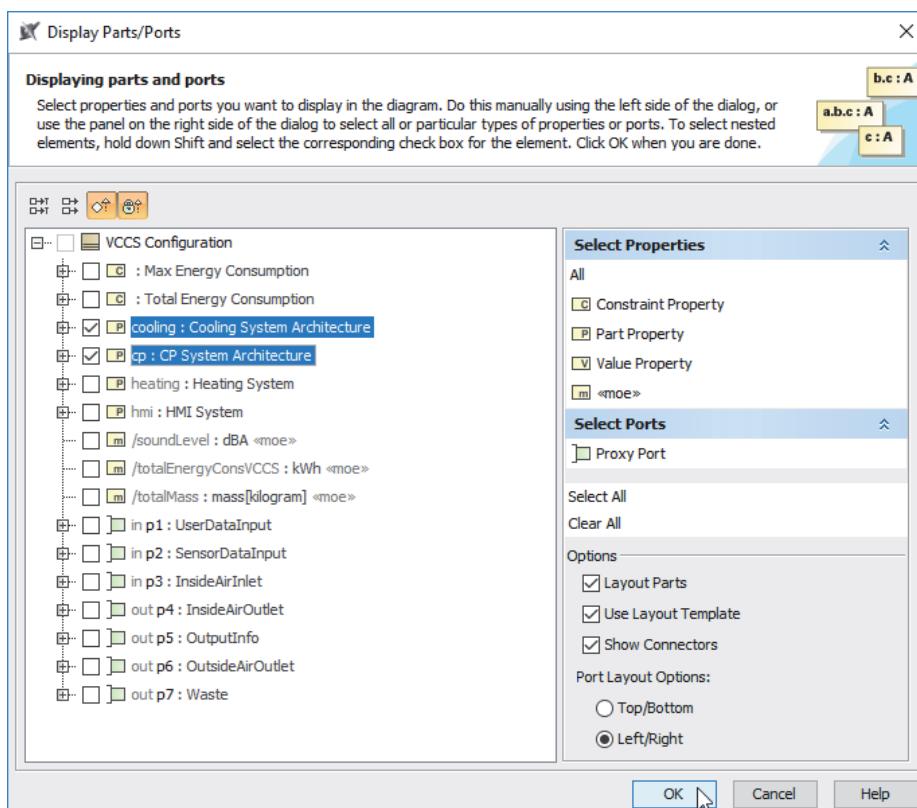
Remember that communication is possible only if these logical subsystems have compatible interfaces. For this reason, we can state that the systems architect performs the interface compatibility verification by specifying the interactions within the particular configuration of the **Sol**.

You already know that interactions between the internal parts of the system can be specified in the SysML ibd. So let's create one and check whether the Cooling System and CP System, two logical subsystems of the VCCS, can successfully communicate.

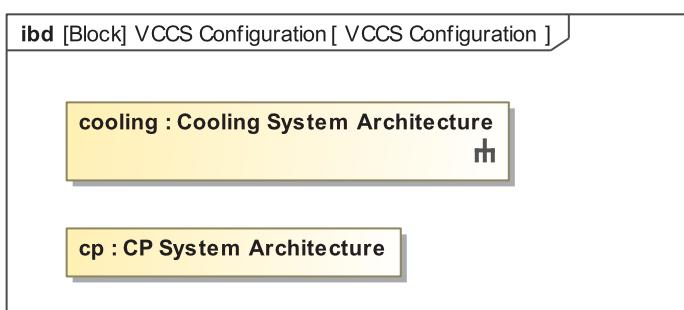
To specify interactions between the Cooling and CP Systems

1. Create an ibd for specifying interactions between the logical subsystems:

- a. In the *VCCS Configuration Structure* diagram you created in [step 2](#) of this cell tutorial, select the shape of the *VCCS Configuration* block.
- b. Click the SysML Internal Block Diagram button  on its smart manipulator toolbar.
- c. A new blank diagram is created.
- d. In the **Display Parts/Ports** dialog, do the following:
 - i. Click **Clear All** on the right, to unselect all the items.
 - ii. Click to select the check boxes near the *cooling : Cooling System Architecture* and *cp : CP System Architecture* part properties (see the following figure) and then click **OK**.



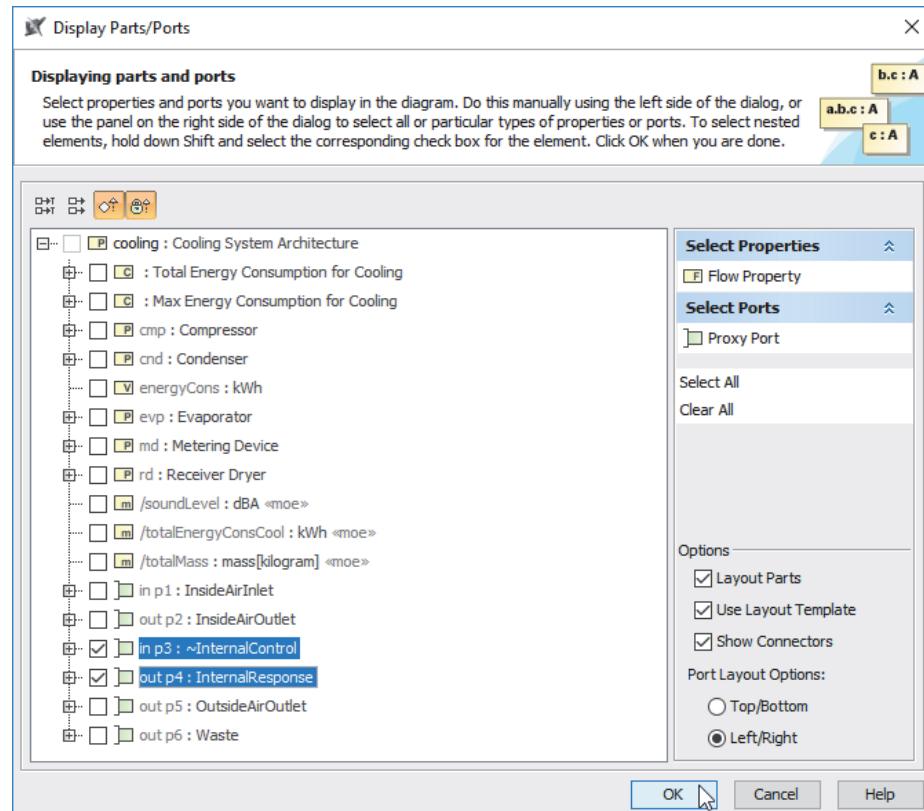
The selected part properties display on the diagram pane.



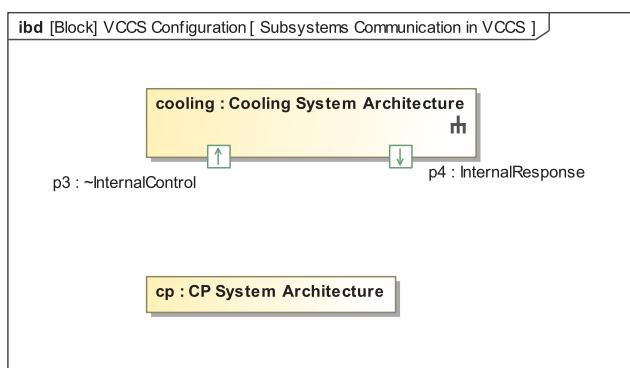
- e. In the Model Browser, select the newly created diagram and press F2 to enter the name edit mode.
- f. Type *Subsystems Communication in VCCS* to change its name and press Enter.

2. Display proxy ports for interactions between the Cooling System and the CP System:

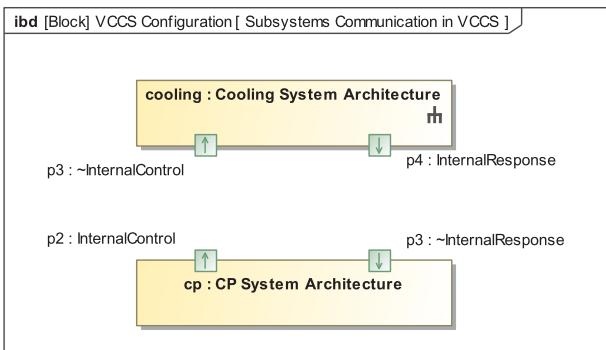
- a. Right-click the shape of the *cooling* property and select **Display > Display Parts/Ports**.
- b. In the **Display Parts/Ports** dialog, do the following:
 - i. Click **Clear All** on the right, to unselect all the items.
 - ii. Click to select the check boxes near the *p3 : ~InternalControl* and *p4 : InternalResponse* proxy ports (see the following figure), and then click **OK**.



The selected proxy ports display on the shape of the *cooling* part property.



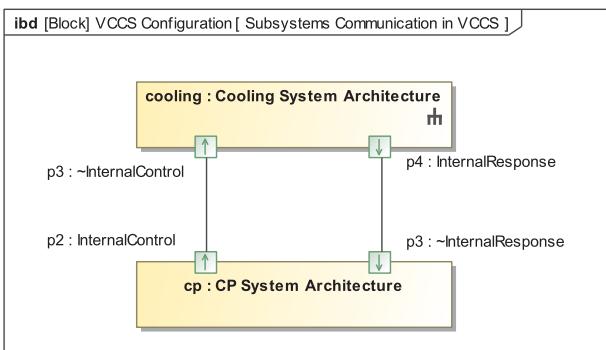
- c. Follow steps a and b to display the relevant proxy ports on the shape of the *cp* part property.



i As you can see, there are two pairs of compatible proxy ports: one pair typed by the *InternalControl* interface block, and the other by the *InternalResponse* interface block. Each pair has one proxy port for data input, and one for data output.

3. Create connectors between the part properties, over compatible proxy ports:

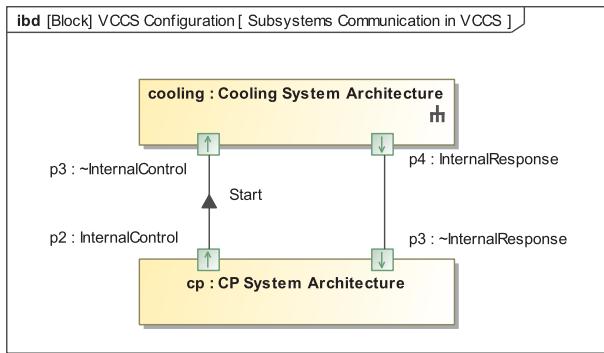
- Select the *p2* proxy port on the shape of the *cp* part property and click the Connector button on its smart manipulator toolbar.
- Click the *p3* proxy port on the shape of the *cooling* part property.
- Follow steps a and b to create a connector between the other pair of compatible proxy ports.



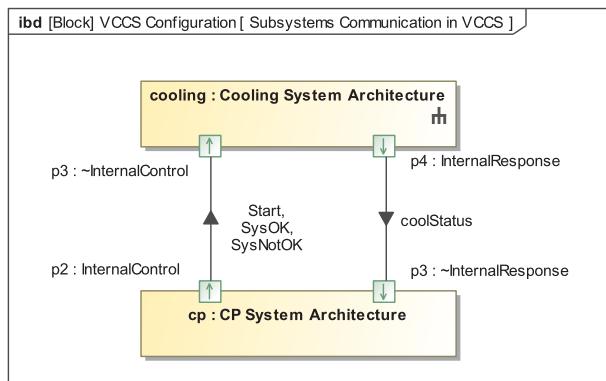
4. Specify the *Start* signal as the item flowing over the connector between the *cp* and *cooling* part properties:

- In the Model Browser, select the *Start* signal:
 - Press Ctrl + Alt + F. The **Quick Find** dialog opens.
 - Type *start* and when you see the *Start* signal selected in the search results list below, press Enter. The *Start* signal is selected in the Model Browser, under the following structure of packages: 2 Solution Domain > 2 System Structure > 2 Exchange Items.
- Drag the selection onto the connector between two proxy ports typed by the *InternalControl* interface block.

c. In the open dialog, don't change anything and click **Finish**.

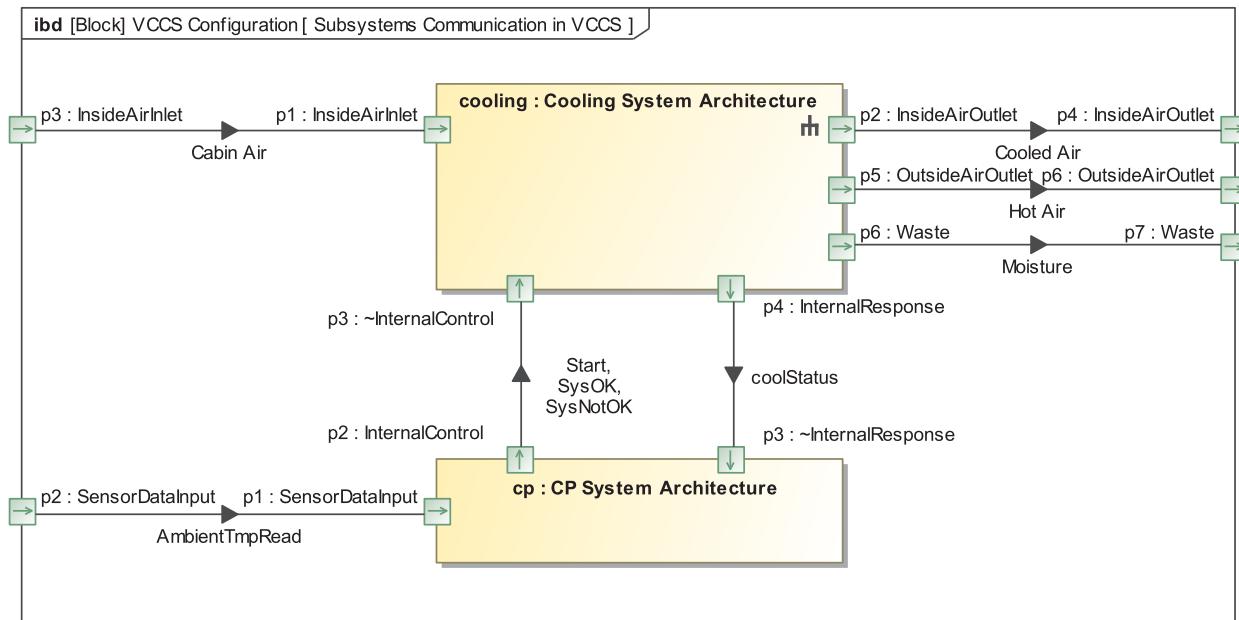


5. Follow step 4 to specify the *SysOK* and *SysNotOK* signals as items flowing over the same connector, and the *coolStatus* signal as the item flowing over the connector between *cooling* and *cp* part properties.



As you can see, no incompatible interfaces were identified, and we can state that the Cooling System and CP System can successfully communicate within the VCCS.

After you specify the interactions of the logical subsystems with the outside, your *Subsystems Communication in VCCS* diagram should look like the one below.



System Configuration Structure done. What's next?

- The integrated model of the system configuration enables you to calculate more precise values of system parameters than in the LSA model. Chapter [System Configuration Parameters](#) describes how to do this.
- Moreover, the specification of logical subsystems communication enables their behavioral models to exchange signals (so far, it was possible to send and receive signals only within the scope of a single logical subsystem). Therefore, you can switch to analyzing the integrated behavior of the whole [Sol](#), which is described in Chapter [System Configuration Behavior](#).

System Configuration Behavior

What is it?

The behavior of the **system of interest** integrates the behaviors of its logical subsystems, identified in the LSA model (see Chapter **System Structure**).

Once you build the integrated structure model of the entire **Sol** (see previous chapter), you have its integrated behavior model as well. No modeling effort is required in this cell. Its main purpose is to analyze the integrated system behavior model and verify it.

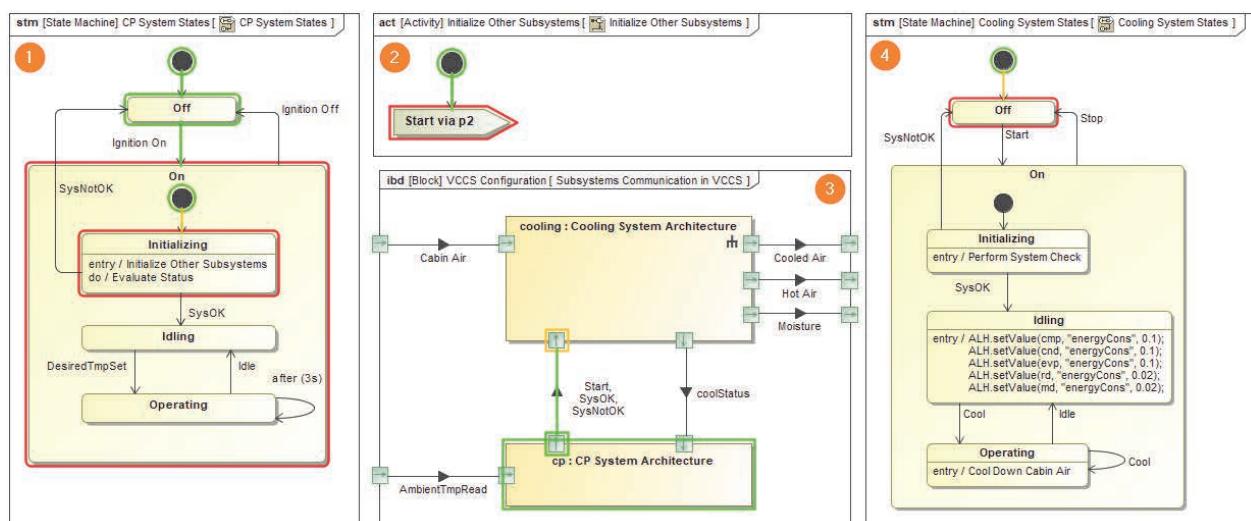
Who is responsible?

The integrated behavior of the system configuration can be analyzed and verified by the Systems Architect or Systems Engineer, who is the head of the whole systems engineering project and is responsible for the smooth integration of the subsystems and their components into the whole, the same person who owns the LSA model of the **Sol**. This task can also be performed by the Systems Integrator, a subordinate of the Systems Architect.

How to model?

Remember that this cell is not about modeling. It is about integrated model analysis. For this, the simulation capabilities of the **modeling tool** can be exploited. By executing the model, you can simulate the behavior of the whole **Sol** and verify if it is as expected.

The simulation animation enables you to monitor the execution progress. It marks the current, last visited, and all other visited element shapes. The following figure depicts the animated moment of model execution, when the *CP System Architecture* block sends out the *Start* signal via the *p2* proxy port and the *Cooling System Architecture* block is about to receive it. Note that the *Subsystems Communication in VCCS* diagram helps; it specifies where the *Start* signal is sent.



Tutorial

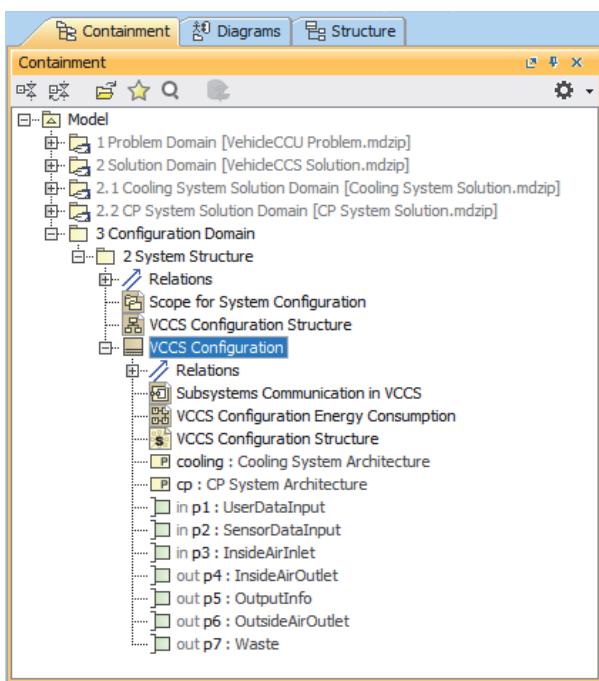
- Step 1. Starting simulation of the system configuration block
- Step 2. Analyzing the integrated behavior of the system configuration

Step 1. Starting simulation of the system configuration block

Let's say you, as the systems architect, perform the analysis of the integrated system behavior to verify the *SR-1.2.1 Internal Communication*, *SSR-1.2.2 Initializing*, and *SSR-1.2.3 Idling* requirements. You will need to prove that the specified system behavior meets the following logic:

1. Once ignition starts, the CP System initializes all other subsystems of the VCCS (in this case, the Cooling System only).
2. In response, the Cooling System performs the self-check and informs the CP System if it can start.
3. After the CP System receives the status from the Cooling System, it decides whether or not they can both start.

To simulate the behavior of the whole configuration of the **Sol**, a relevant block must be indicated as the simulation context. In this case, it is the *VCCS Configuration* block, shown in the following figure.

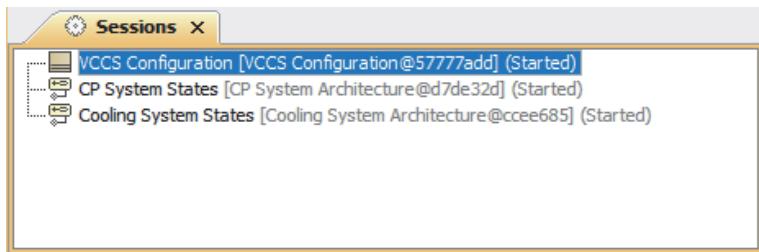


- i** If the system behavior has been defined in LSA model and it is still assigned as the classifier behavior of the *VCCS* block (the super-type of the *VCCS Configuration* block), during the simulation it will conflict with the integrated behavior of the *VCCS Configuration* block.

To prevent this conflict, go back to the LSA model and set the **Classifier Behavior** property value to **<UNASSIGNED>** for the *VCCS* block (refer to Chapter [System Behavior](#)). If you cannot update the LSA model (for example, you don't have relevant permission), you can define a fake system behavior for the *VCCS Configuration* block (such as the state machine with a single default state). As a result, the fake system behavior becomes the classifier behavior, which doesn't conflict with the integrated one during the simulation.

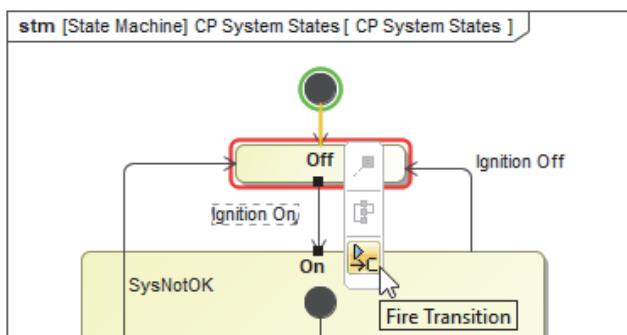
To start the simulation of the VCCS Configuration block

1. In the Model Browser, select the *VCCS Configuration* block.
2. Right-click the selection and then select **Simulation > Run**. The **Simulation** panel opens at the bottom of the **modeling tool** window.
3. Click the Auto Open Diagrams button  to enable the mode, in which currently executed diagrams open automatically.
4. On the toolbar of the **Simulation** panel, click the Start button  . The simulation sessions are created for the state machines of each block that is a part of the *VCCS Configuration* block. In this case, these are the state machines of the *Cooling System Architecture* and *CP System Architecture* blocks.

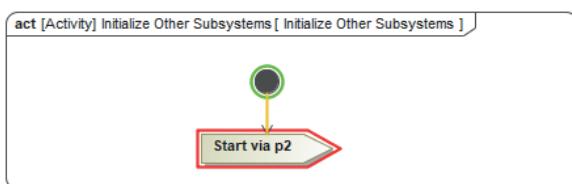


Step 2. Analyzing the integrated behavior of the system configuration

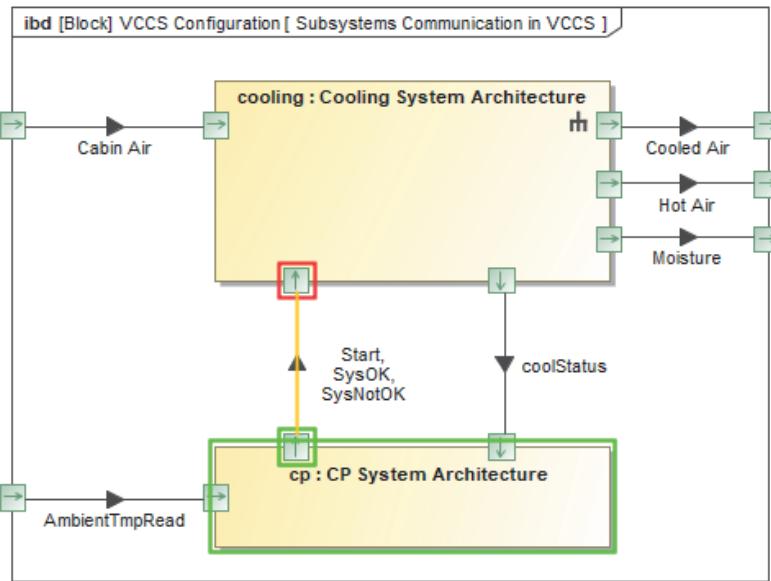
When the model execution begins, monitor the simulation progress until both state machines rest in the *Off* states. The following figure shows what happens when you fire the transition between the *Off* and *On* states in the *CP System States* diagram.



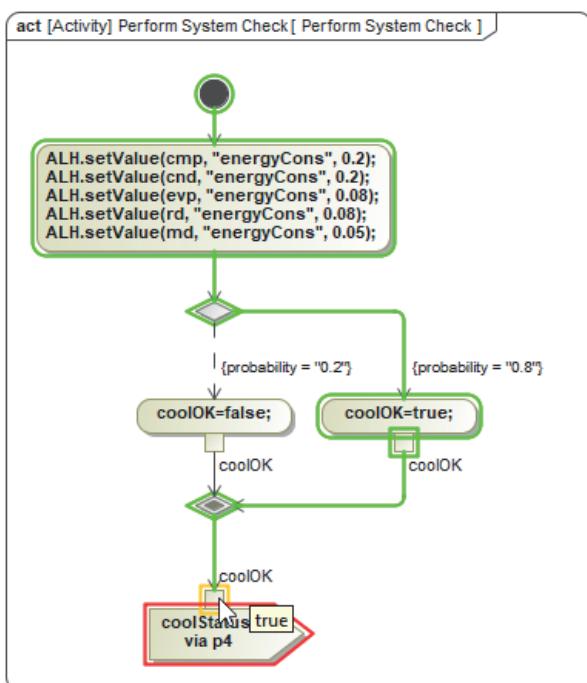
As a result of the manually fired transition, the *CP System Architecture* block enters the *Initializing* state. On entering that state, it sends out the *Start* signal via the *p2* proxy port (see the following figure).



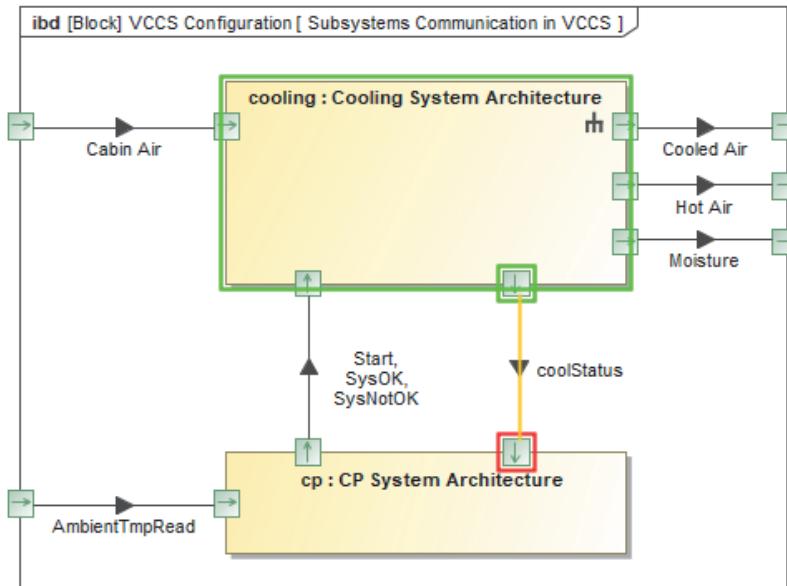
As the *Subsystems Communication in VCCS* diagram defines, the *Start* signal is received by the *Cooling System Architecture* block.



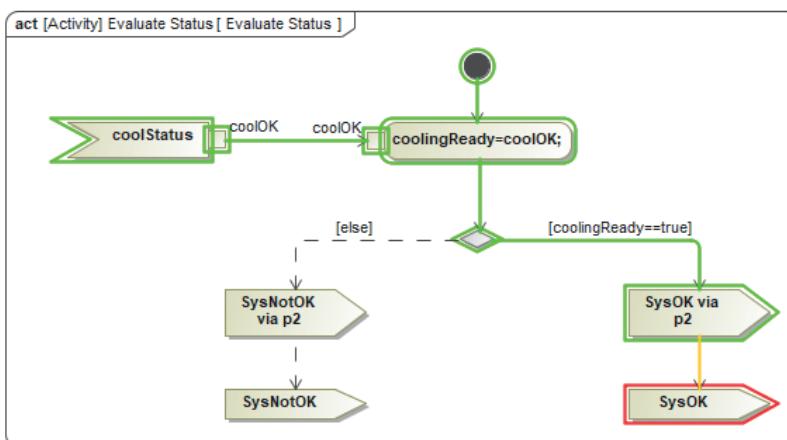
Acceptance of the *Start* signal automatically fires the transition between the *Off* and *On* states displayed in the *Cooling System States* diagram. As a result, the *Cooling System Architecture* block enters the *Initializing* state. On entering the *Initializing* state, it performs the *Perform System Check* activity and sends out the *coolStatus* signal via the *p4* proxy port. Remember, the *coolStatus* signal has the *coolOK* attribute with the *true* or *false* value.



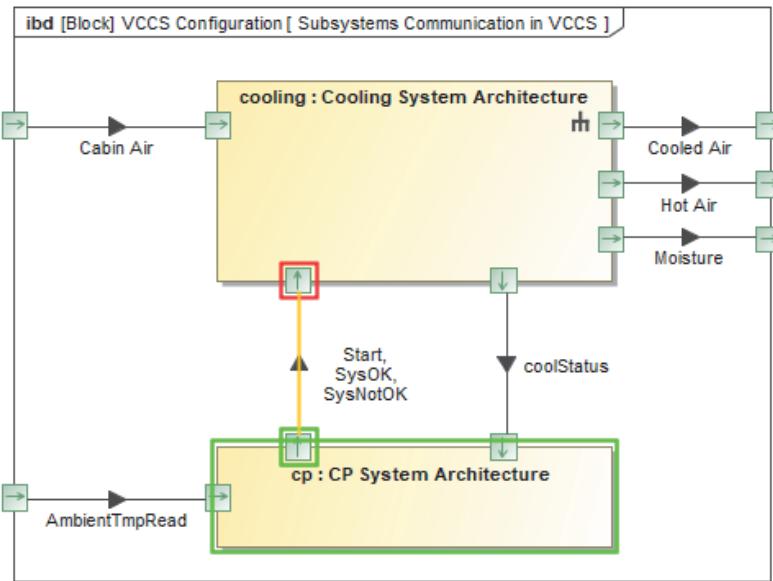
As the *Subsystems Communication in VCCS* diagram defines, the *coolStatus* signal is received by the *CP System Architecture* block.



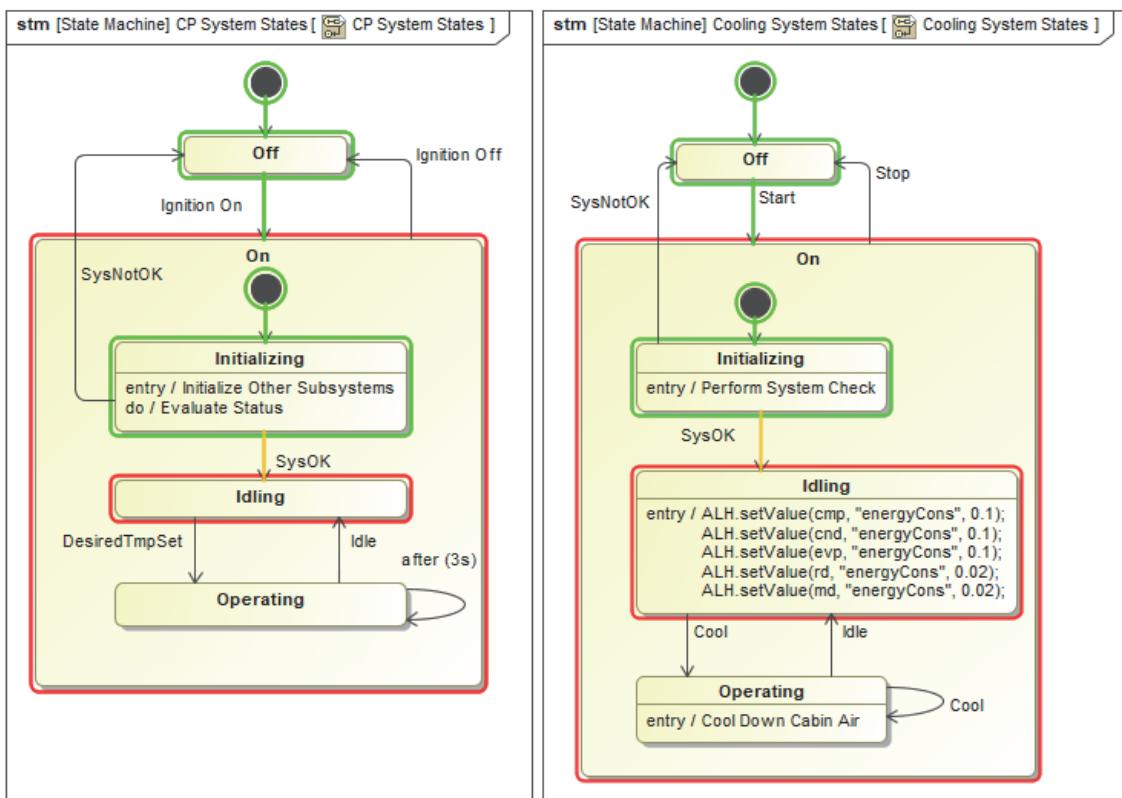
The *CP System Architecture* block accepts the *coolOK* signal, reads its attribute value, and decides whether or not both subsystems can switch to the *Idling* state. For this, it performs the *Evaluate Status* activity.



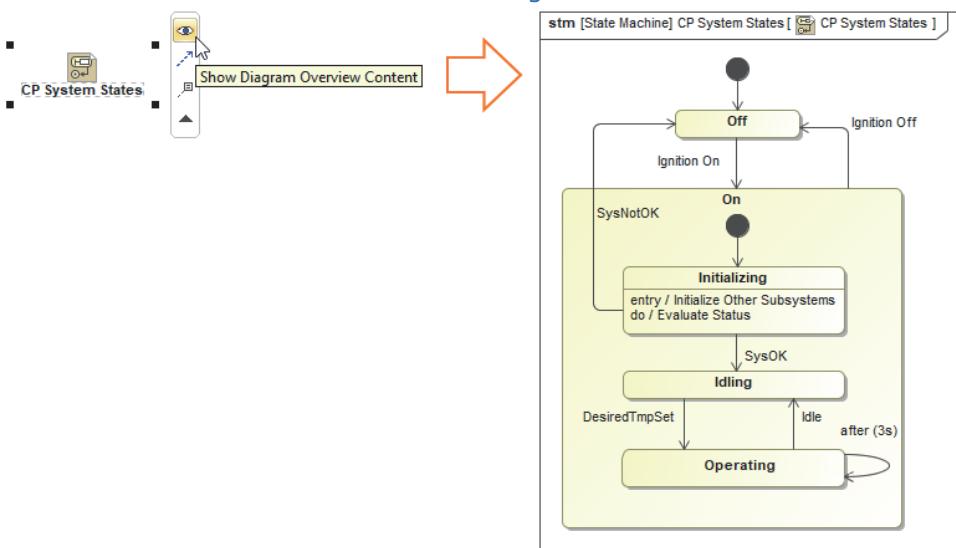
As a result, the *CP System Architecture* block sends out either the *SysOK* or *SysNotOK* signal via the *p2* proxy port (see the following figure). As the figure shows, the same signal in each case is also sent internally, this time to fire one of the transitions of the *CP System Architecture* block: either to the *Idling* or to the *Off* state.



As a result, both blocks capturing the logical subsystems move to the *Idling* states of their state machines (see the following figure). Here they wait for the *DesiredTmpSet* signal to proceed.



i If you want to display two or more diagrams on a single diagram pane, create a content diagram, drag all the diagrams you want to see to its pane from the Model Browser, and convert each diagram shape into the diagram overview shape (see the following figure). For more information, see the latest documentation of the [modeling tool](#).



This analysis proves that the integrated system behavior model meets the relevant system and subsystem requirements.

System Configuration Behavior done. What's next?

After you finish the analysis of the integrated system behavior model, you can switch to the last task of [Building system configuration model](#) and calculate more precise values of system parameters than in the LSA model. Chapter [System Configuration Parameters](#) describes how to do this.

System Configuration Parameters

What is it?

The system configuration model enables you to relate the calculation methods from different levels of detail, such as one from the system with some from the subsystem level. You can define more specific input values and get more precise calculation results than in the LSA model. For example, in the LSA model, you had to specify the energy consumption of the Cooling System, while in the system configuration model it can be derived by summing up the energy consumption of its logical components.

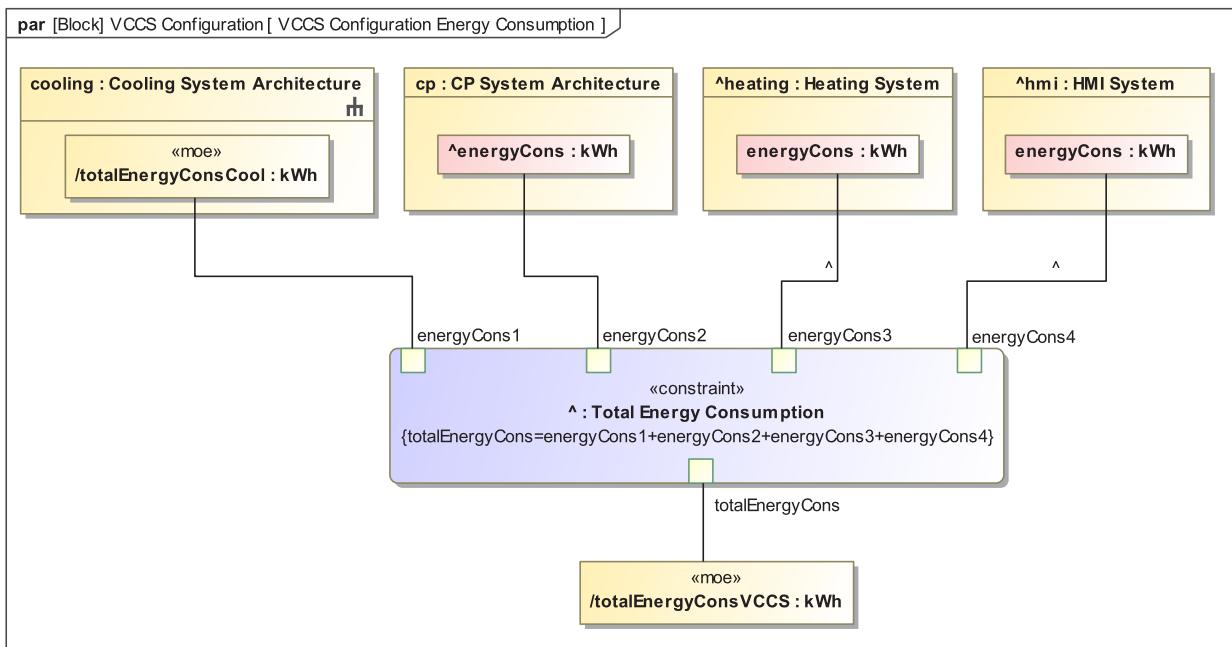
Who is responsible?

The system configuration parameters can be analyzed and verified by the Systems Architect or Systems Engineer, who is the head of the entire systems engineering project and is responsible for the smooth integration of the subsystems and their components into the whole; the same person who owns the LSA model of the [Sol](#). This task can also be performed by the Systems Integrator, a subordinate of the Systems Architect.

How to model?

Usually you don't need to capture any calculation method in the system configuration model. You've done that in the LSA model and also in the LSSA models. You've also defined the system, subsystem, and component parameters there. You don't need to re-capture them in the system configuration model as well.

However, you may need to update the bindings. For example, you might need to bind a constraint parameter specified in the LSA model to the value property created in the LSSA model. In this case, the infrastructure of the SysML parametric diagram helps a lot.



Tutorial

- [Step 1. Redefining the MoE](#)
- [Step 2. Creating a parametric diagram](#)
- [Step 3. Updating bindings](#)

Step 1. Redefining the MoE

Let's say you want to check how the energy consumption of the logical components of the Cooling System affects the total energy consumption of the whole VCCS. For this, you need to do a few things.

First of all, redefine the *totalEnergyConsumptionVCCS* value property inherited from the *VCCS* block. The redefinition is not necessary for calculations, as you can perform them with an inherited value property. However, you need to redefine the value property to be able to create a satisfy relationship from the value property to the relevant system requirement.

To redefine the *totalEnergyConsumptionVCCS* value property

1. Open the *VCCS Configuration Structure* diagram you created in [step 2 of this cell tutorial](#).
2. Right-click the *VCCS Configuration* block and select **Specification**.
3. On the left of the open dialog, click **Properties**.
4. Under the **Value Properties** category, select the *totalEnergyConsumptionVCCS* value property click the **Redefine** button below.
5. As a result, the selected value property is redefined and is now no longer decorated with the caret symbol ("^"), which indicates the inheritance. Don't change anything.

Properties				
	Name	Type	Default Value	Owner
Constraint Properties				
^	Total Energy Cons...		VCCS [2 Solution D...	
^	Max Energy Consu...		VCCS [2 Solution D...	
Part Properties				
cooling	Cooling System Arc...		VCCS Configurat...	
cp	CP System Archite...		VCCS Configurat...	
^ hmi	HMI System [2 Sol...		VCCS [2 Solution D...	
^ heating	Heating System [2 ...		VCCS [2 Solution D...	
Value Properties				
	totalEnergyConsVCCS	v kWh [2 Solution Do...	VCCS Configurat...	
^ soundLevel	dBA [1 Problem Do...		MoEs Holder [1 Pro...	
^ totalMass	mass[kilogram] [IS...		MoEs Holder [1 Pro...	

6. Click the **Close** button.

Now you're ready for the next step.

Step 2. Creating a parametric diagram

The next thing to do is relate the value property capturing the total energy consumption of the Cooling System to the input parameter of the constraint property, which defines the method for calculating the total energy consumption of the whole VCCS.

Creating a SysML parametric diagram could be a good start to establish this relationship. Remember, this type of diagram can only be created under a block. This time, it is the *VCCS Configuration* block.

To create a SysML parametric diagram for the *VCCS Configuration* block

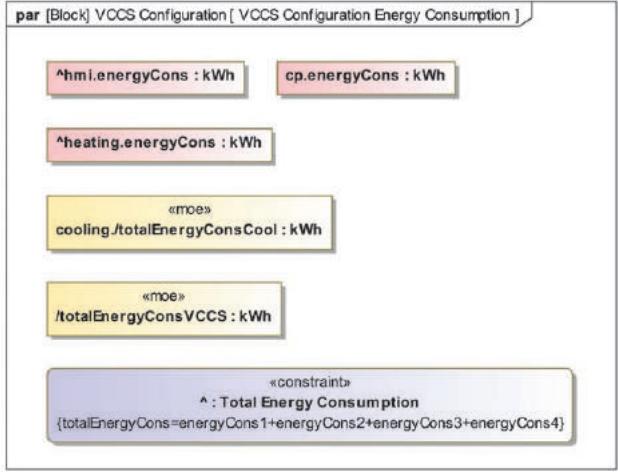
1. In the Model Browser, right-click the block and select **Create Diagram**.
2. Select **SysML Parametric Diagram** in the **Create Diagram** dialog below. A new diagram is created, and the **Display Parameters/Properties** dialog is opened.
3. In the dialog:
 - a. Leave the check box near the *:Total Energy Consumption* constraint property selected.

(i) This constraint property defines the method for calculating the total energy consumption of the VCCS. You already specified it for the *VCCS* block in the LSA model. It belongs to the *VCCS Configuration* block, too, because it is a sub-type of the *VCCS* block and inherits all its features from the *VCCS* block, including the constraint properties.
 - b. Click the button  near the *:Cooling System Architecture* part property to expand its contents.
 - c. Click to select the check box near the */totalEnergyConsCool : kWh* value property.

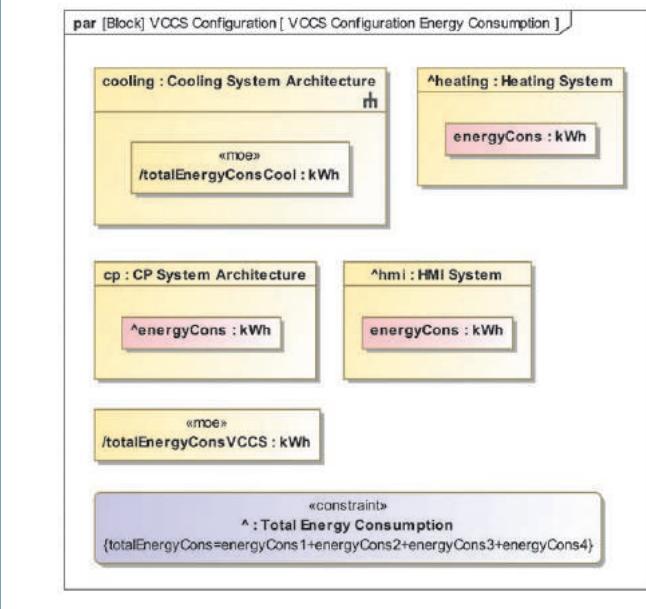
(i) This subsystem parameter is derived from component parameters, by summing up the energy consumed by the Compressor, Condenser, Evaporator, etc. When calculating the total energy consumption of the VCCS, you need to replace the *energyCons : kWh* value property specified in the LSA model with this value property.
 - d. Click the button  near the *:CP System Architecture* part property to expand its contents.
 - e. Click to select the check box near the *energyCons : kWh* value property.

(i) As you don't have yet the method defining the energy consumption of the CP System, you can take the value property defined in the LSA model. The same logic applies for other logical subsystems.
 - f. Follow steps d and e to select the *energyCons : kWh* value properties of the *:Heating System* and *:HMI System* part properties.
 - g. Make sure the check box near the */totalEnergyConsVCCS : kWh* value property is selected.
 - h. Make sure everything else is unselected.
 - i. Click **OK** to close the dialog.
4. In the Model Browser, type *VCCS Configuration Energy Consumption* to name the newly created diagram.

Once you're done, the *VCCS Configuration Energy Consumption* diagram should look like the following figure.



- i** If you want to, you can change the dot notation of displaying value properties to the nested one. For this, select all the shapes, except the one of the *totalEnergyConsVCCS* value property, then right-click them and select **Refactor > Convert to Nested Parts**.

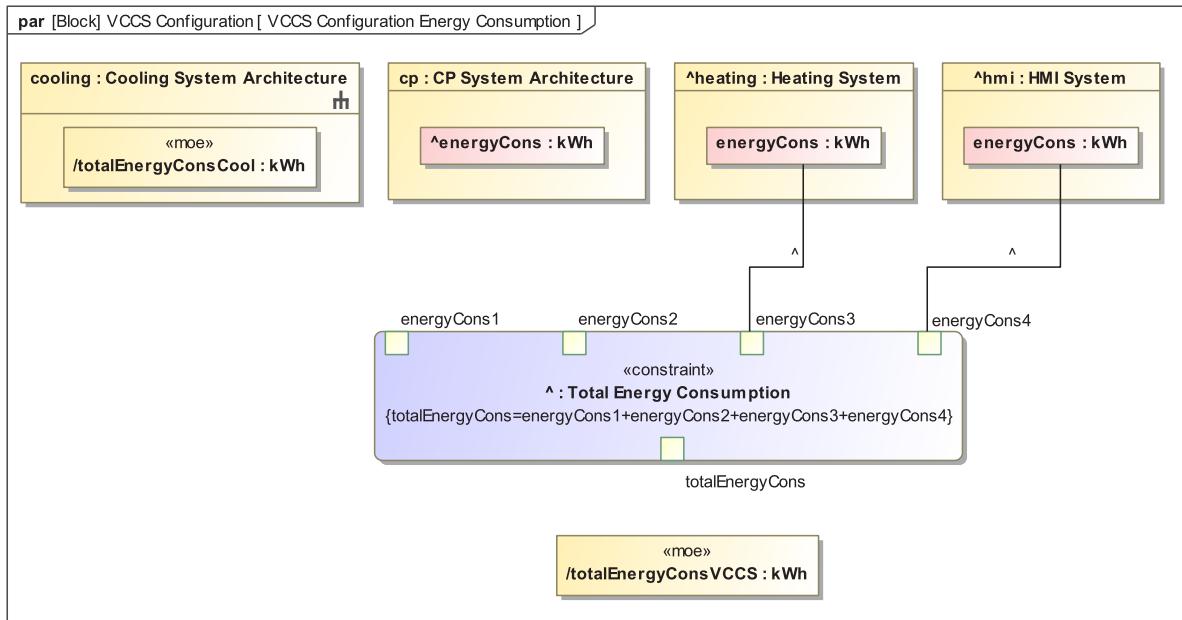


Step 3. Updating bindings

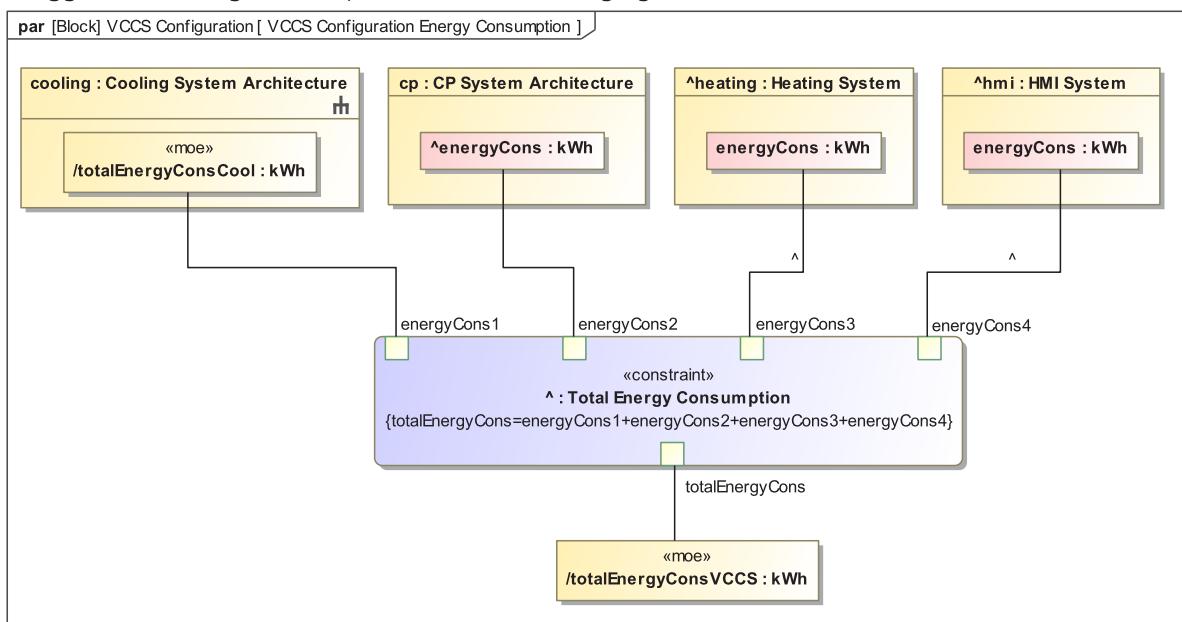
Now you can update the bindings and re-run the calculations.

To update the bindings in the parametric diagram

1. Select the constraint property and click the Display All Parameters button on its smart manipulator toolbar. Parameters and binding connectors to corresponding value properties are displayed on the diagram.



2. Select the shape of the `/totalEnergyConsCool : kWh` value property within the shape of the `cooling : Cooling System Architecture` part property.
3. Click the Binding Connector button on its smart manipulator toolbar.
4. Click the shape of the `energyCons1` parameter. The new binding connector is created, and the parametric diagram is ready to calculate the total energy consumption of the VCCS encountering the energy consumed by the components of the Cooling System.



5. Repeat steps 2 to 4 to establish another binding connector between the `CP System Architecture` block and the `energyCons2` parameter.
6. Repeat steps 2 to 4 to establish one more binding connector, this time between the `/totalEnergyConsVCCS` value property of the `VCCS Configuration` block and the `totalEnergyCons` parameter.

Now, if you run the simulation on this diagram, you can enter diverse values of energy consumption for subsystems and components, and see the result. The result values can be stored in the model (see [step 7](#) in the [System Parameters](#) tutorial).

Name	Value
VCCS Configuration	VCCS Configuration@1013735e
m /soundLevel : dBA	0,0000
m /totalEnergyConsVCCS : kWh	3,4600
m /totalMass : mass[kilogram]	0,0000
P cooling : Cooling System Architecture	Cooling System Architecture@6caeef93
V energyCons : kWh	1,1400
m /soundLevel : dBA	0,0000
m /totalEnergyConsCool : kWh	1,1400
m /totalMass : mass[kilogram]	0,0000
P cmp : Compressor	Compressor@31840d64
V energyCons : kWh	0,3200
P cnd : Condenser	Condenser@6032e220
V energyCons : kWh	0,2200
P evp : Evaporator	Evaporator@5aa0fcdb
V energyCons : kWh	0,5000
P md : Metering Device	Metering Device@4f78a568
V energyCons : kWh	0,0500
P rd : Receiver Dryer	Receiver Dryer@54a3d45d
V energyCons : kWh	0,0500
P cp : CP System Architecture	CP System Architecture@117f1ae8
V coolingReady : Boolean	<input type="checkbox"/> false
V energyCons : kWh	1,0500
P heating : Heating System	Heating System@3bc6e657
V energyCons : kWh	0,6500
P hmi : HMI System	HMI System@14225c2e
V energyCons : kWh	0,6200

When you have the solution architectures of all logical subsystems integrated into the system configuration model, you can update other bindings in this diagram.

System Configuration Parameters done. What's next?

Finishing the system configuration parameters means that you're almost done with the system configuration model. Only traceability relationships to system requirements specification are left to deal with.

Traceability from System Configuration to System Requirements

What is it?

To ensure that the system configuration model is complete, and all system requirements are considered and implemented, you need to perform the analysis and specify traceability relationships from the system configuration model to the system requirements specification. These are satisfy relationships.

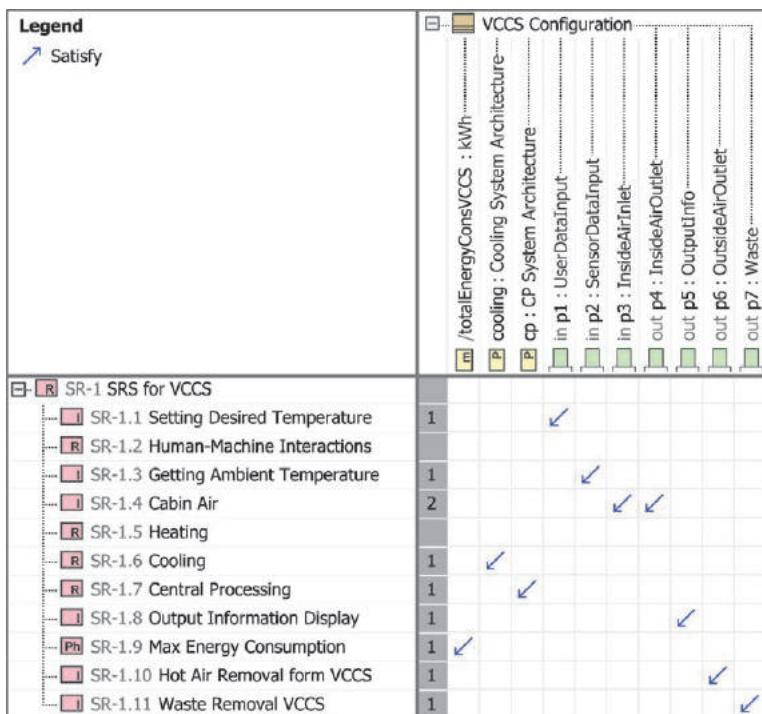
Who is responsible?

Traceability relationships can be specified by the Systems Architect or Systems Engineer, who is the head of the entire systems engineering project and is responsible for the smooth integration of the subsystems and their components into the whole.

How to model?

As you already know, a Satisfy Requirement Matrix provides the best infrastructure for specifying satisfy relationships. The matrix is pretty much the same as the one created in the LSA model. Only the column scope is different: here you should select the *VCCS Configuration* block instead of the *VCCS* block.

The matrix in the following figure shows that a few system requirements are not yet satisfied. For this, you need to build the LSSA of the Heating and HMI Systems and then integrate them into the system configuration model.



Tutorial

Steps for specifying the traceability relationships are equivalent to steps of the [Traceability to System Requirements](#) tutorial:

- [Step 1. Creating a matrix for capturing satisfy relationships](#)
- [Step 2. Capturing satisfy relationships](#)

Solution Domain done. What's next?

Having the complete system configuration model means that you're done with the entire [Solution domain](#), which indicates that you're ready to move to the [Implementation domain](#) and ready to start considering your system real.

IMPLEMENTATION DOMAIN

After the logical architecture and design of the whole system is defined and the optimal system configuration is selected (by means of the trade-off analysis), the system is ready for implementation. At this point, you should think of the system as real, and not abstract, as you did with 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 framework only defines how to specify implementation requirements for the **Sol**. However, detailed physical design of the **Sol** is not a part the MagicGrid framework. The next steps of the **Sol** development can be accomplished by using such software as electrical and mechanical CAD tools or automated code generation tools, among others.

		Pillar				
Domain	Problem	Requirements	Structure	Behavior	Parameters	Safety & Reliability
		Stakeholder Needs	System Context	Use Cases	Measures of Effectiveness (MoEs)	Conceptual and Functional Failure Mode & Effects Analysis (FMEA)
			Conceptual Subsystems	Functional Analysis	MoEs for Subsystems	Conceptual Subsystems FMEA
	Solution	System Requirements	System Structure	System Behavior	System Parameters	System Safety & Reliability (S&R)
		Subsystem Requirements	Subsystem Structure	Subsystem Behavior	Subsystem Parameters	Subsystem S&R
		Component Requirements	Component Structure	Component Behavior	Component Parameters	Component S&R
	Implementation	Implementation Requirements				

Implementation Requirements

What is it?

In order to implement the system of interest, you first need to have detailed requirements to follow while producing the system's detailed physical design. These requirements are based on the logical architecture of the **system of interest**, including the logical architectures of all its subsystems and even more precise units.

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

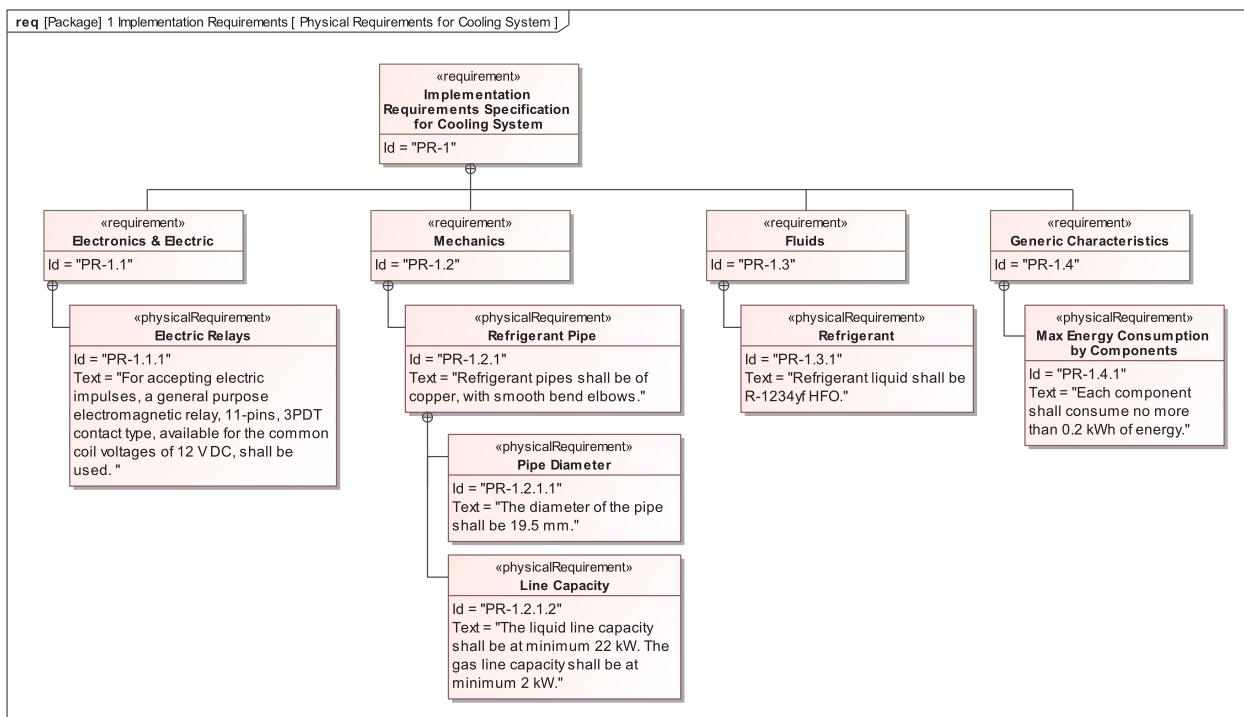
Who is responsible?

Implementation requirements specification can be produced by the Systems Designer.

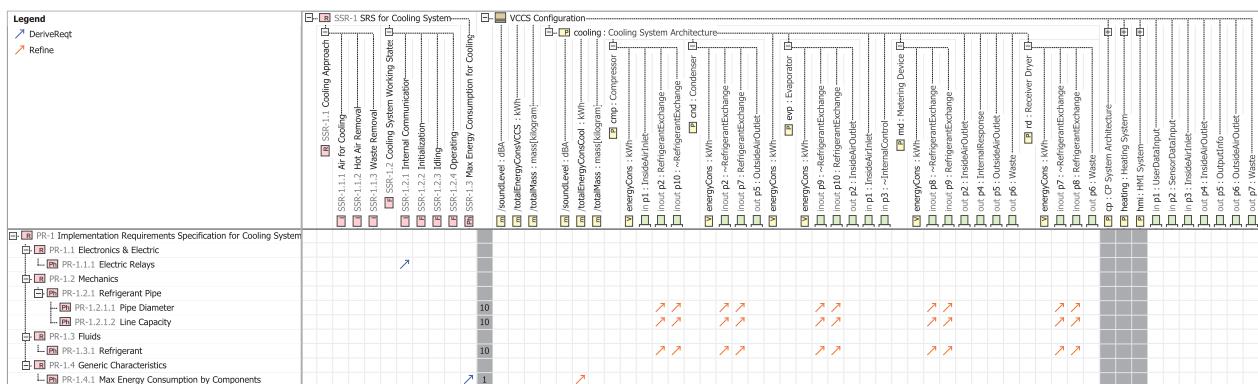
How to model?

In the modeling tool, an item of the implementation requirements specification can be stored as a SysML requirement, which has a unique identification, name, and textual specification. Implementation requirements must be formal and written down by following certain guidelines. These may include writing in short and clear sentences only; avoiding conditional keywords, such as *if, except, as well*; use of the *shall* keyword; and incorporating tables. For more information on writing good textual requirements, refer to the *INCOSE Systems Engineering Handbook*.

Managing implementation requirements is similar to managing system, subsystem, and component requirements. As with those requirements, implementation requirements can be categorized and organized into groups, sub-groups, and so forth. SysML supports a wide variety of requirement categories, such as physical, functional, and performance, etc. For grouping requirements, use the containment relationships among them. The SysML requirements that capture grouping items might not contain any text. As you can see in the following figure, which shows an example of implementation requirements for the Cooling System, there are four groups of requirements: *Electronics & Electric*, *Mechanics*, *Fluids*, and *Generic Characteristics*.

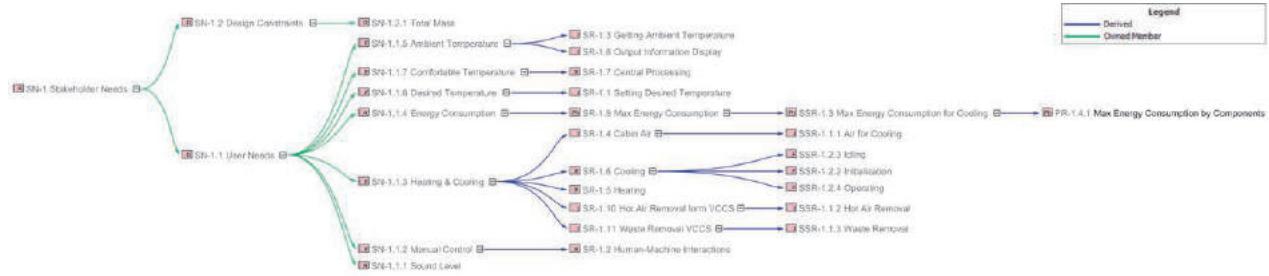


As shown in the following figure, each implementation requirement should refine one or more elements from the system configuration model, or be derived from at least one requirement from the solution domain. For this, a matrix for specifying both types of relationships can be created (similar to the one in [step 4 of Subsystem Requirements](#)).



As you can conclude by analyzing the matrix, the implementation requirements specification is not yet completed. It will be considered to be complete when each element within the system configuration model is refined by one or more implementation requirements and each solution domain requirement is covered by at least one implementation requirement.

Having derivation relationships among requirements in different domains, you can easily trace from very detailed requirements to very abstract requirements and see what items determine the creation of other items. To perform this 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.



Tutorial

Steps of this cell tutorial are adequate to steps of the [Subsystem Requirements](#) tutorial:

- [Step 1. Creating and organizing a model for subsystem requirements](#)
- [Step 2. Creating a diagram for subsystem requirements](#)
- [Step 3. Specifying subsystem requirements](#)
- [Step 4. Specifying traceability relationships](#)

The difference is that in step 4 you establish traceability relationships to the subsystem requirements for the Cooling System and to the relevant elements of the system configuration model.

What's next?

You are ready to develop the detailed physical design of the system. These activities are outside the scope of the MagicGrid approach.

ANNEX A: SAFETY & RELIABILITY ANALYSIS

By Andrius Armonas, PhD, Elona Paulikaitytė, Osvaldas Jankauskas, Tomas Juknevičius

Reliability is the ability of a functional unit to perform a required function under given conditions for a given time interval (ISO/IEC 2382:2015 Information Technology). Safety is freedom from unacceptable risk (IEC 61508:2010 EEEPE safety-related systems).

SysML, as one of the key MBSE components, has a good foundation for capturing requirements, architecture, constraints, views, and viewpoints. However, SysML does not provide the necessary constructs to capture safety and reliability information in the system model. The Cameo Safety and Reliability Analyzer plugin adds support for those missing constructs. The following safety and reliability standards are supported:

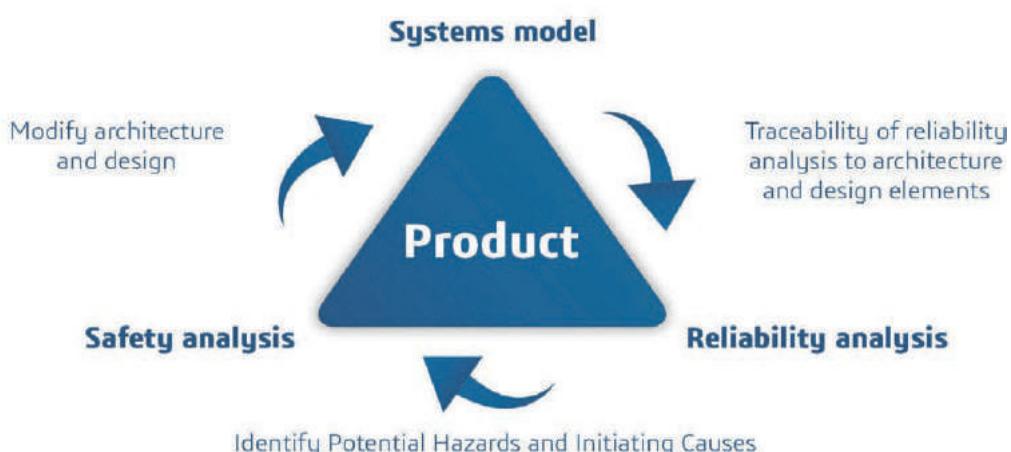
- The failure mode, effects, and criticality analysis (FMEA) according to IEC 60812:2006 standard
- Hazard analysis according to the following medical standards:
 - IEC 62304
 - ISO 14971:2007, Corrected version 2007-10-01 (Medical devices - Application of risk management to medical devices)

The ISO 26262 Functional Safety plugin, used in conjunction with the Cameo Safety and Reliability Analyzer plugin, supports ISO 26262 (Road vehicles – Functional safety) standard.

This annex describes how FMEA, supported by the Cameo Safety and Reliability Analyzer plugin, is used to enrich a systems model created using the MagicGrid method with safety and reliability information.

The figure below depicts the general safety and reliability analysis workflow. Reliability and safety analysis is performed against the systems model (stakeholder needs, system behavior, and structural parts) and identified as separate items, which are called FMEA items when the FMEA method is used. These items are then ranked and the most critical are addressed by introducing new safety and reliability requirements. Subsequently, changes to the architecture and design of a system are done by introducing new behaviors and altering the structure of the system to support the identified safety and reliability requirements. Newly introduced modifications to the systems model need to undergo the same safety and reliability assessment.

Safety and reliability analysis is either performed by a safety engineer or a systems engineer.



Stakeholder needs: safety and reliability requirements

Safety and reliability requirements, as identified by stakeholders, can be captured as SysML requirements. See Chapter [Stakeholder Needs](#) for more information on how to capture stakeholder needs in requirement tables. Capturing safety and reliability requirements using SysML requirements does not require any extensions of the SysML language; a grouping requirement with a corresponding name is sufficient. As you can see in the following figure, the *SN-1.3 Safety & Reliability* requirement acts as a container for safety and reliability requirements that originate from stakeholders.

#	Name	Text
1	□ R SN-1 Stakeholder Needs	
2	□ R SN-1.1 User Needs	
3	R SN-1.1.1 Sound Level	Climate control unit in max mode shall not be louder than engine.
4	F SN-1.1.2 Manual Control	I should be able to start and stop climate control by myself.
5	F SN-1.1.3 Heating & Cooling	The unit must be able to heat and cool.
6	R SN-1.1.4 Energy Consumption	I'd like the unit to consume as little energy as possible.
7	F SN-1.1.5 Ambient Temperature	I want to see the ambient temperature on the screen or some other output device.
8	F SN-1.1.6 Desired Temperature	It should be a possibility to easily specify the desired temperature.
9	F SN-1.1.7 Comfortable Temperature	I'd like to feel comfortable temperature while being in the cabin.
10	□ R SN-1.2 Design Constraints	
11	R SN-1.2.1 Total Mass	Mass of the unit shall not exceed 2 percent of the total car mass.
12	□ R SN-1.3 Safety & Reliability	
13	R SN-1.3.1 Heat air to the desired temperature in 5 minutes	Heat air to the desired temperature in 5 minutes.
14	□ R SN-1.3.2 Harm to passenger	
15	R SN-1.3.2.1 Resistance to fire	Climate control unit will not cause fire on its own and will not add to fire started from other causes.
16	R SN-1.3.2.2 Biofouling	Passengers of a car should not be exposed to any toxic materials accumulated in climate control unit.

Conceptual FMEA in Black Box view

The Black Box view should contain qualitative conceptual component and functional failure mode analysis based on the use of the FMEA method. The purpose of the conceptual FMEA is to take actions to eliminate or reduce failures that originate at the Black Box view. The reduction of failure is achieved by introducing new safety and reliability requirements as a result of the conceptual FMEA analysis and then refining and satisfying these requirements with the changes downstream.

Traditional FMEAs are done in a spreadsheet-like environment, supported by the Cameo Safety and Reliability Analyzer plugin as a primary way to manage FMEAs. Each row in the table is called an FMEA item which references (or semantically groups) classical FMEA elements, such as failure cause, failure mode, local and final effects, and mitigation.

The Black Box conceptual FMEA table contains the following columns:

Column name	Purpose
Id	A unique ID assigned to each FMEA Item.
Name	A user-friendly name of an FMEA Item.
Item	The part undergoing analysis related to a particular FMEA Item.
Cause of Failure	An element indicating the architecture weakness causing a Failure Mode. An FMEA Item, e.g., a single row of an FMEA Table, can have multiple Causes of Failure.
Failure Mode	An element describing the specific manner in which a system could potentially fail to meet the design intent.

Column name	Purpose
Local Effect of Failure	An element describing the effect that a Failure Mode has on the system element under consideration. An FMEA Item, e.g., a single row of an FMEA Table, can have multiple Local Effects of Failure.
Final Effect of Failure	An element describing the effect that Failure Mode has on an end-user or environment. You can specify multiple Final Effect of Failure values for a single FMEA Item.
Refines	The reference to a stakeholder requirement which the FMEA Item refines.
Mitigation	The reference to a requirement that mitigates a failure.

The following figure shows an FMEA table supported by the Cameo Safety and Reliability Analyzer plugin, which represents the conceptual FMEA. The item is the `:VCCU` unit, which is the system being modeled. Every FMEA item refines safety and reliability requirements from stakeholder needs. For example, the `F-2` FMEA item refines the `SN-1.3.2.2 Biofouling` stakeholder requirement. The refining FMEA item identifies how the `:VCCU` unit can fail to meet the need for passengers of a car to not be exposed to any toxic materials accumulated in the climate control unit:

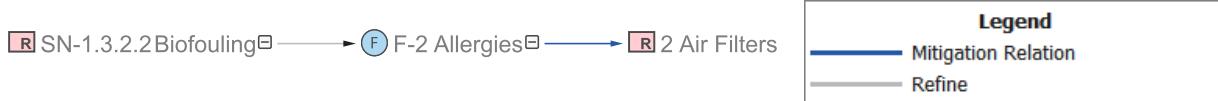
- First, potential causes of failure are identified; in this particular case, direct contact of a passenger with toxic materials accumulated in the climate control unit.
- Then, failure modes are identified; in this particular case, passengers not being able to operate the `:VCCU` unit (since the unit produces toxic materials).
- Local and final effects of failure are either introduced or reused from the library as they tend to repeat across product lines or different products.
- Finally, new safety or reliability requirements are introduced; in this particular case, a requirement to have air filters so that toxic materials would not accumulate in the climate control unit.

#	△ Id	Name	Item	Cause Of Failure	Failure Mode	Local Effect Of Failure	Final Effect Of Failure	Refines	Mitigation
1	F-1	VCCU on fire due to internal fault	: VCCU		VCCU severely overheated	Fire spreads to other systems Emit smoke VCCU not operational Loss of containment	Burns from fire Poisoning from smoke Direct death from fire Accident while driving	SN-1.3.2.1 Resistance to fire	1 Use Flame-Resistant Materials
2	F-2	Allergies	: VCCU	Direct contact of a passenger with toxic materials accumulated in climate control unit.	VCCU unit cannot be operated by a passenger	Emit allergens	Discomfort while operating VCCU Allergic reactions affecting skin or pulmonary system	SN-1.3.2.2 Biofouling	2 Air Filters
3	F-3	Insufficient heating	: vccu	Insufficient heating power Big difference between ambient and desired temperature	Reduction of function	VCCU not being able to reach required temperature in time	Passengers in uncomfortable temperature due to insufficient heating	SN-1.3.1 Heat air to the desired temperature in 5 minutes	3 Provide Auxiliary Heating

As already mentioned, FMEA items can have a name. The name is a resumptive sentence that conveys the essence of analysis performed at concrete FMEA items. The purpose of the FMEA item name is to optimize the search of FMEA items in the model to promote the reuse and facilitate the management of FMEA items.

Causes may not be specified at all if for any reason or cause within the created system a failure mode occurs and only that fact is important to introduce a mitigation safety or reliability requirement (see the `F-1` FMEA item). The newly introduced safety and reliability requirements then need to be linked to the newly created or updated behavioral or structural models in both Black Box, White Box, LSA, Subsystem, or Component views.

Safety and reliability analysis workflow is repetitive, thus failures that originate at the Black Box view can resonate through the whole Sol, from the problem domain down to implementation. In order to capture the failure scenarios, FMEA cause-effect chains are used that can be depicted using Relation Map diagrams. For example, the *Biofouling* Relation Map (see the following figure) depicts the model elements and FMEA items that originate during FMEA analysis starting from the `SN-1.3.2.2 Biofouling` stakeholder requirement.



This Relation Map is going to be expanded gradually during the FMEA analysis whenever the failure effects (of newly introduced changes in the system architecture) are identified leading to an unintended scenario.

Functional FMEA in Black Box view

The functional FMEA focuses on behaviors of the system rather than refining stakeholder requirements. The Black Box functional FMEA table contains the following additional columns comparing with conceptual FMEA table:

Column name	Purpose
Subsystem	incoming and/or outgoing flow connecting action allocated to part defined in the Item column.
Source (<i>derived</i>)	source action of flow.
Target (<i>derived</i>)	target action of flow.

The Black Box functional FMEA is depicted in the following figure. As you can see, the Item column is still referencing :VCCU, just as in the conceptual FMEA. However, the Subsystem column represents object flows that are analyzed for potential failures. The source for this analysis is the activity diagrams specified for use cases (see the [Use Cases](#) section). The *Provide Comfortable Temperature* activity from the [Use Cases](#) section is shown below the functional FMEA table. When performing a functional FMEA, all incoming and outgoing flows into the swimlane representing :VCCU, as well as actions being performed by it, are being analyzed for potential failures (e.g., how would the system perform if one of the incoming flows that flows into the given action did not produce the expected output?).

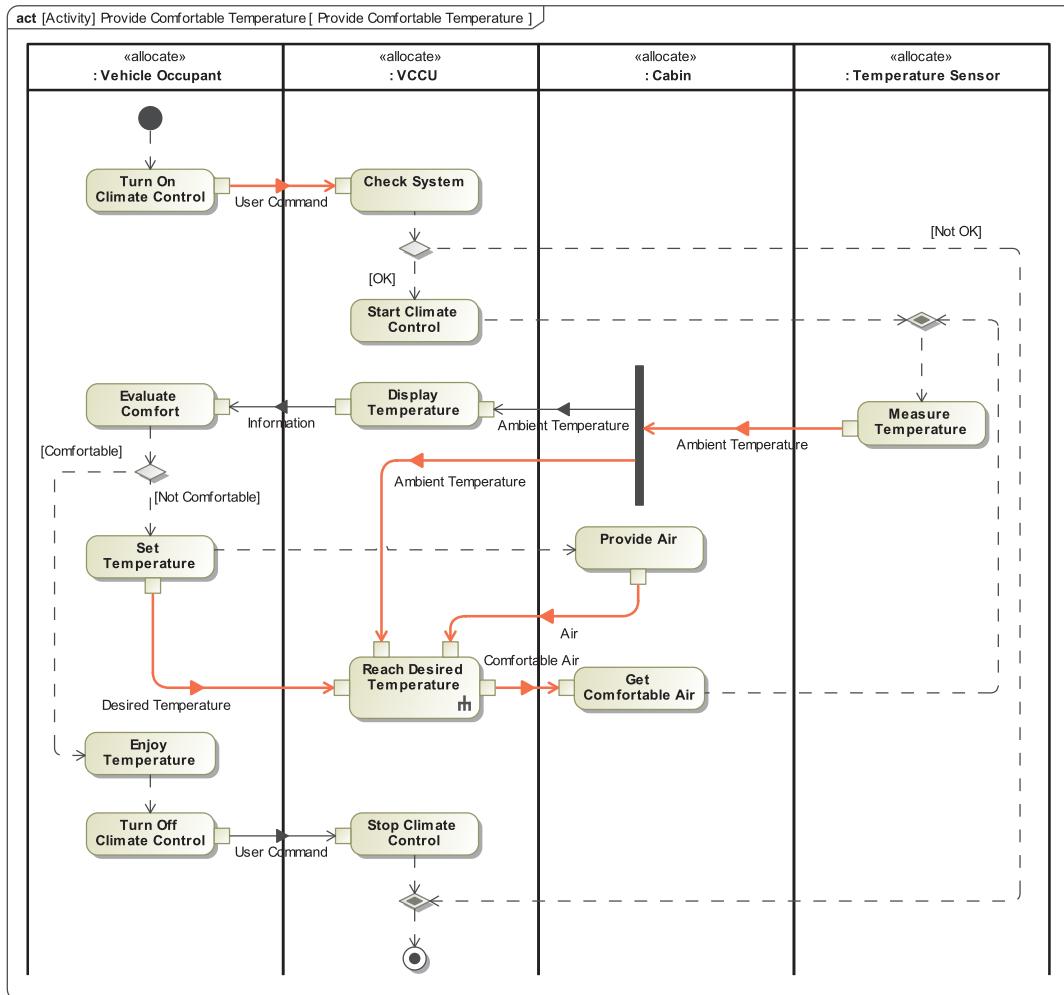
In this particular example, we only focus on object flows related to the *Check System* and *Reach Desired Temperature* actions, but the same can also be done for control flows and actions. The flows being analyzed are color-coded in orange in the activity diagram below the FMEA table. The actions that these flows connect are represented in the Source and Target columns in the functional FMEA table. The Refines column represents the analyzed use case. The Mitigation column has the same purpose as in the conceptual FMEA table.

#	Id	Name	Item	Subsystem	Source	Target	Cause Of Failure	Failure Mode	Local Effect Of Failure	Final Effect Of Failure	Refines	Mitigation
1	F-4	Overheating or undercooling when it cannot be turned on	:VCCU	Object Flow[output ↳ -> input]	:Turn On Climate Control	:Check System	VCCU does not accept the command to start	Loss of function	VCCU not operational	Passengers overheated or undercooled	Provide ○ Comfortable Temperature	4 Provide Limp Mode
2	F-5	Overheating or undercooling when it does not accept or receive set temperature	:VCCU	Object Flow[-> ↳ Desired Temperature]	:Set Temperature	:Reach Desired Temperature	VCCU does not accept or receive set temperature	Loss of function	VCCU not operational	Passengers overheated or undercooled	Provide ○ Comfortable Temperature	4 Provide Limp Mode
3	F-6	Overheating or undercooling when it does not accept or receive ambient temperature measurement	:VCCU	Object Flow[-> ↳ Ambient Temperature]	:Measure Temperature	:Reach Desired Temperature	VCCU does not accept or receive ambient temperature measurement	Loss of function	VCCU not operational	Passengers overheated or undercooled	Provide ○ Comfortable Temperature	5 Shut Down the System in Fail-Safe Manner when Unable to Read or Accept Ambient Temperature
4	F-7	Overheating or undercooling when it cannot suck air from cabin	:VCCU	Object Flow[output ↳ -> Air]	:Provide Air	:Reach Desired Temperature	VCCU cannot suck air from cabin	Loss of function	VCCU not operational	Passengers overheated or undercooled	Provide ○ Comfortable Temperature	6 Shut Down the System in Fail-Safe Manner when Unable to Suck Air from Cabin
5	F-8	Overheating or undercooling when it cannot blow conditioned air into cabin	:VCCU	Object Flow[Comfortable Air -> input]	:Get Comfortable Air	:Reach Desired Temperature	VCCU cannot blow conditioned air into cabin	Loss of function	VCCU not operational	Passengers overheated or undercooled	Provide ○ Comfortable Temperature	7 Shut Down the System in Fail-Safe Manner when Unable to Blow Conditioned Air into Cabin

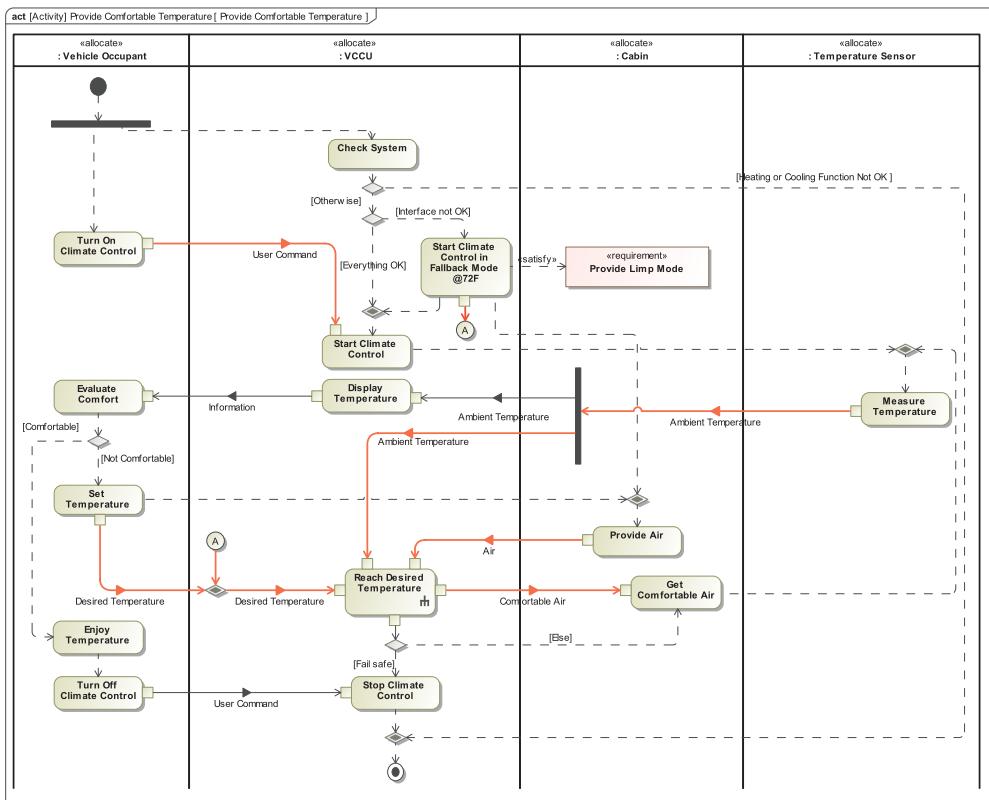
The F-7 FMEA item analyzes the object flow coming from the *Provide Air* into the *Reach Desired Temperature* action by assessing the possible impact of the failure to receive air to the system when it is trying to reach the desired temperature, which is captured as the cause in the *Cause of Failure* column in the FMEA table (*VCCU cannot suck air from cabin*). As a result, the failure mode *Loss of function* occurs,

resulting in the local effect being *VCCU not operational* and the final effect on passengers being overheated or undercooled. To mitigate this risk, a new safety requirement for automatically shutting the system down in fail-safe mode is introduced (see the requirement 6). As in the conceptual FMEA, the newly introduced requirements are addressed either at the problem level or downstream. Other functional FMEA items are analyzed in a similar fashion, demonstrated with the F-7 FMEA item.

To demonstrate how safety and reliability requirements are addressed at this level or downstream, let's have a look at the newly introduced *Provide Limp Mode* safety requirement. The original MagicGrid activity diagram for the *Provide Comfortable Temperature* is shown as follows.



To address the *Provide Limp Mode* safety requirement, the new *:VCCU* function called *Start Climate Control in Fallback Mode @72F* is introduced. This mode would provide the required default 72 °F temperature as long as the *:VCCU* has not yet accepted the command to start or received the desired temperature.



Functional FMEA coverage analysis

It is important to cover all incoming and outgoing flows to the `:VCCU` unit with the functional FMEA; thus, a traceability matrix should be built (see the following figure). In this matrix it is clearly visible how many of the flows still need to be covered by FMEA items.

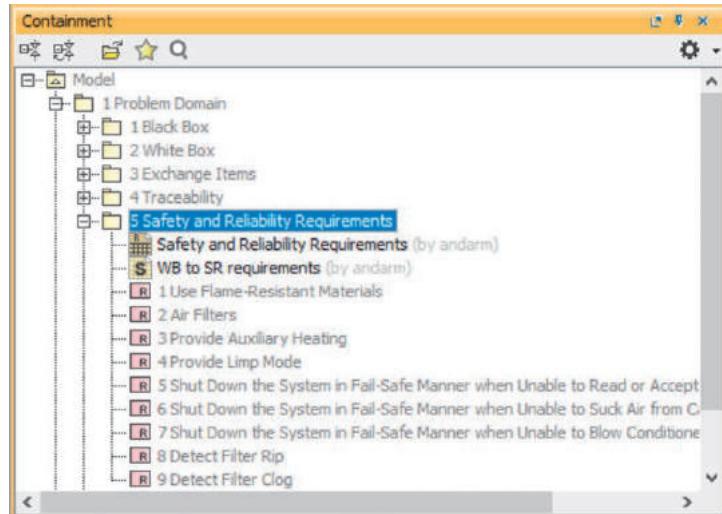
Legend

- Subsystem Relation

	Functional Failures							
	F-4 Overheating or un...	F-5 Overheating or un...	F-6 Overheating or un...	F-7 Overheating or un...	F-8 Overheating or un...	F-9 Overheating or un...	F-10 Overheating or un...	F-11 Overheating or un...
<code>:P :VCCU</code>	1	1	1	1	1	1	1	1
<code>:Check System</code>								
<code>:Display Temperature</code>								
<code>:Object Flow[-> input]</code>								
<code>:Object Flow[output -> input]</code>								
<code>:Reach Desired Temperature</code>								
<code>:Object Flow[-> Ambient Temperature]</code>	1							
<code>:Object Flow[-> Desired Temperature]</code>	1							
<code>:Object Flow[Comfortable Air -> input]</code>	1							
<code>:Object Flow[Fail Safe ->]</code>	1							
<code>:Object Flow[output -> Air]</code>	1							
<code>:Start Climate Control</code>								
<code>:Object Flow[output -> input]</code>	1							
<code>:Start Climate Control in Fallback Mode @72F</code>								
<code>:Object Flow[output ->]</code>	1							
<code>:Stop Climate Control</code>								
<code>:Object Flow[output -> input]</code>	1							

Addressing safety and reliability requirements in Problem domain

All newly introduced safety and reliability requirements are contained in one package (see the following figure).

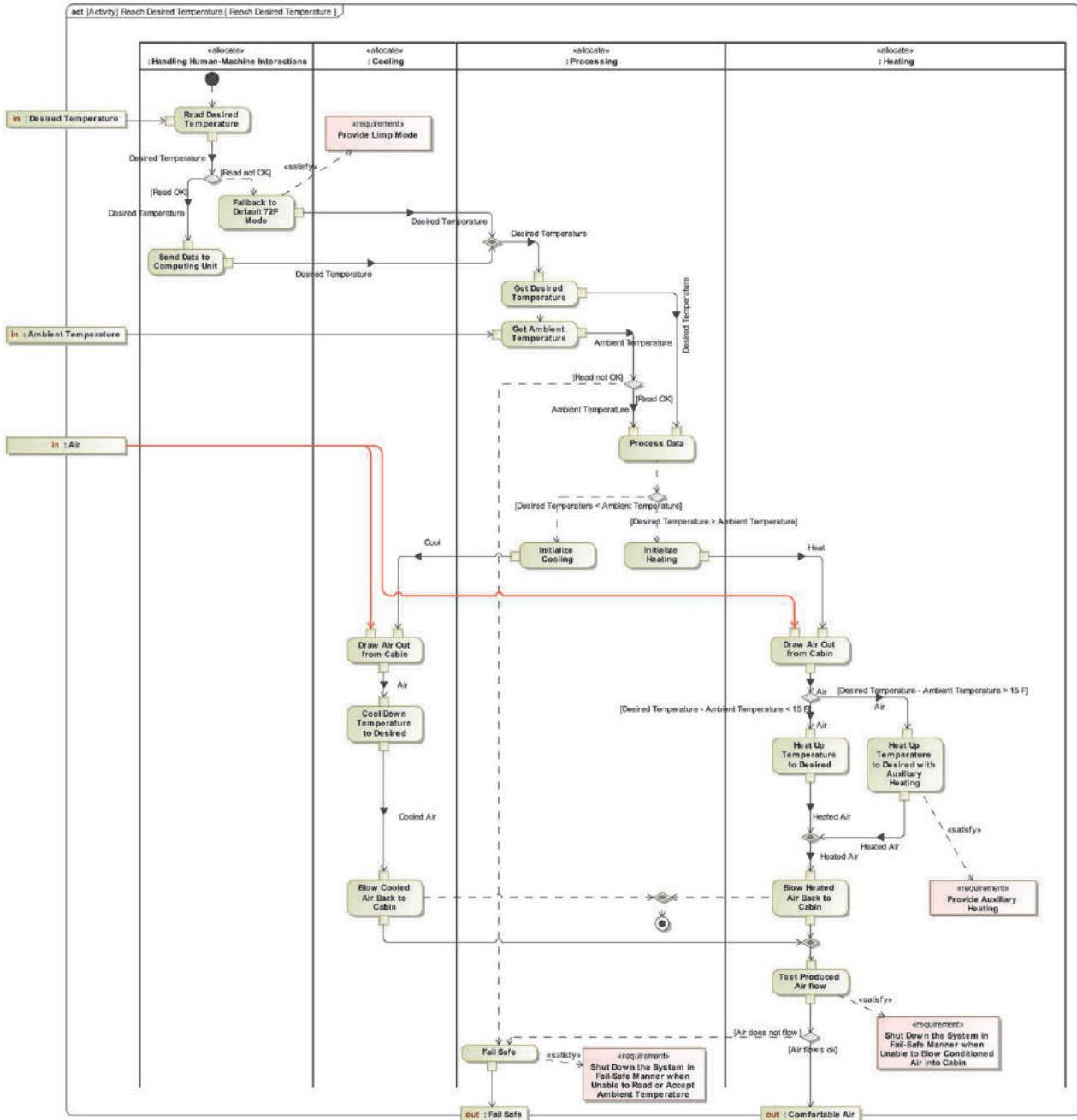


The table below lists safety and reliability requirements. Requirements 8 and 9 are created as a result of the FMEA analysis at the White Box level (see the sections below).

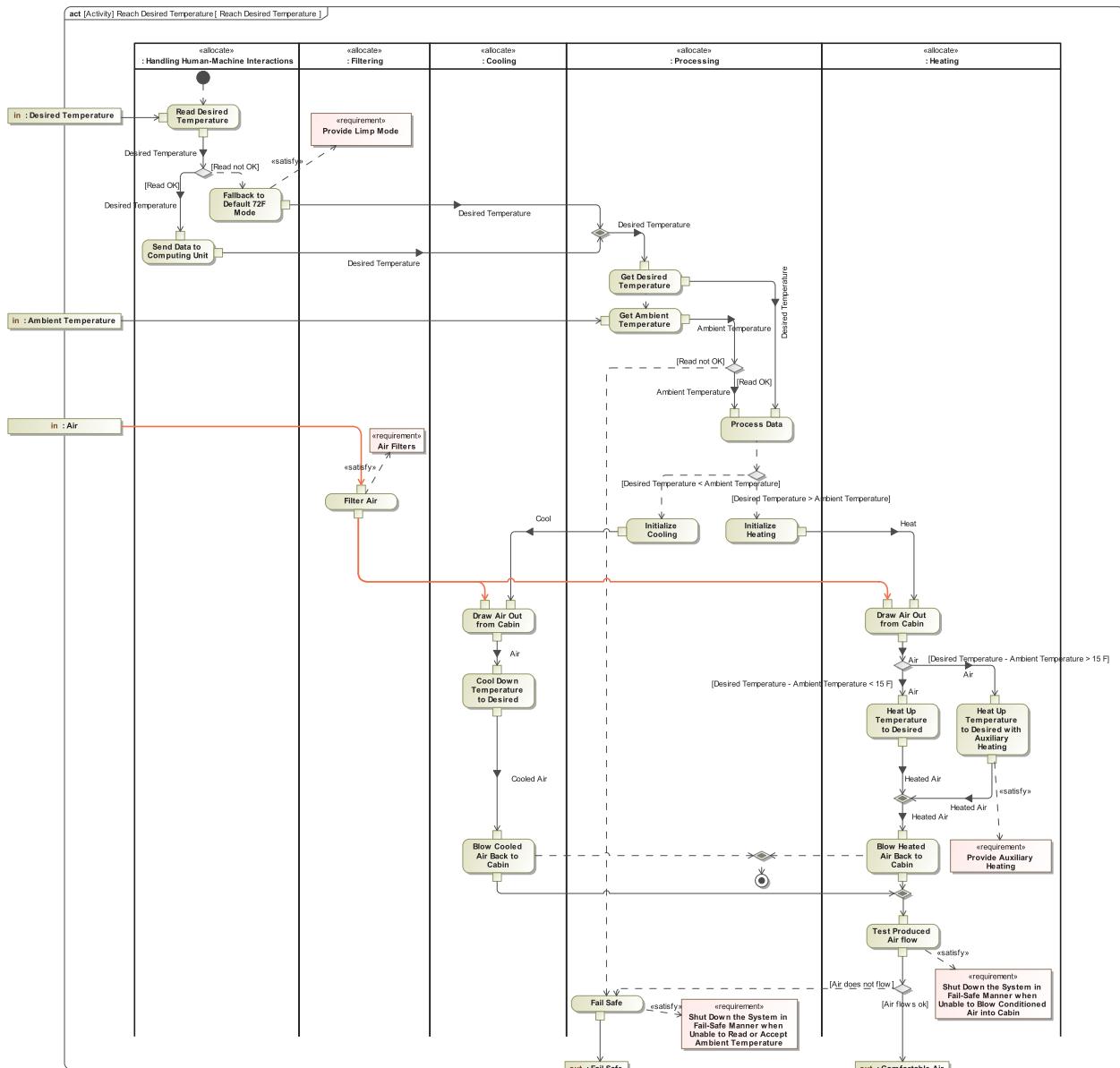
#	△ Name	Text
1	R 1 Use Flame-Resistant Materials	Use materials of HMIS flammability class I or less.
2	R 2 Air Filters	The system should have filters to prevent toxic materials accumulated in the climate control unit from reaching the passenger.
3	R 3 Provide Auxiliary Heating	Provide enough power to heat air to the desired temperature in 5 minutes.
4	R 4 Provide Limp Mode	VCCU shall be able to operate in limp mode by automatically keeping 72F temperature in the cabin.
5	R 5 Shut Down the System in Fail-Safe Manner when Unable to Read or Accept ambient temperature.	Shut down the system in fail-safe manner when unable to read or accept ambient temperature.
6	R 6 Shut Down the System in Fail-Safe Manner when Unable to Suck Air from cabin	Shut down the system in fail-safe manner when unable to suck air from cabin
7	R 7 Shut Down the System in Fail-Safe Manner when Unable to Blow Conditioned air into cabin.	Shut down the system in fail-safe manner when unable to blow conditioned air into cabin.
8	R 8 Detect Filter Rip	Detect Filter Rip.
9	R 9 Detect Filter Clog	Detect Filter Clog.

Safety and reliability requirements can be addressed at all levels, from the problem statement to the component design. For example, the safety requirement *1 Use Flame-Resistant Materials* cannot be addressed at the Problem level since this level does not contain actual system design. This requirement could be addressed at the logical subsystem level since specific materials used in the design of a system are known at this level. Another example is the requirement *2 Air Filters*, which can be addressed at the White Box level and downstream as well.

The original MagicGrid activity diagram for the *Reach Desired Temperature* activity (at the White Box view) is depicted below. The air is flowing directly to the actions for drawing air from the cabin.

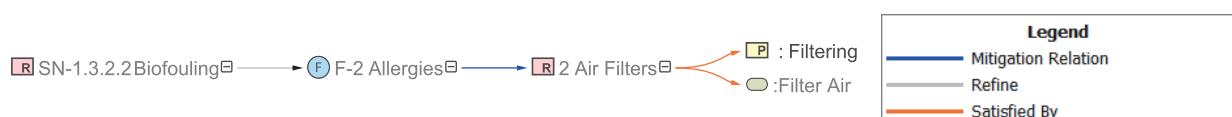


To address the *2 Air Filters* safety requirement, the new *:Filtering* logical subsystem needs to be added with an air filtering function and corresponding flows to *:Cooling* and *:Heating* subsystems (see the following figure).

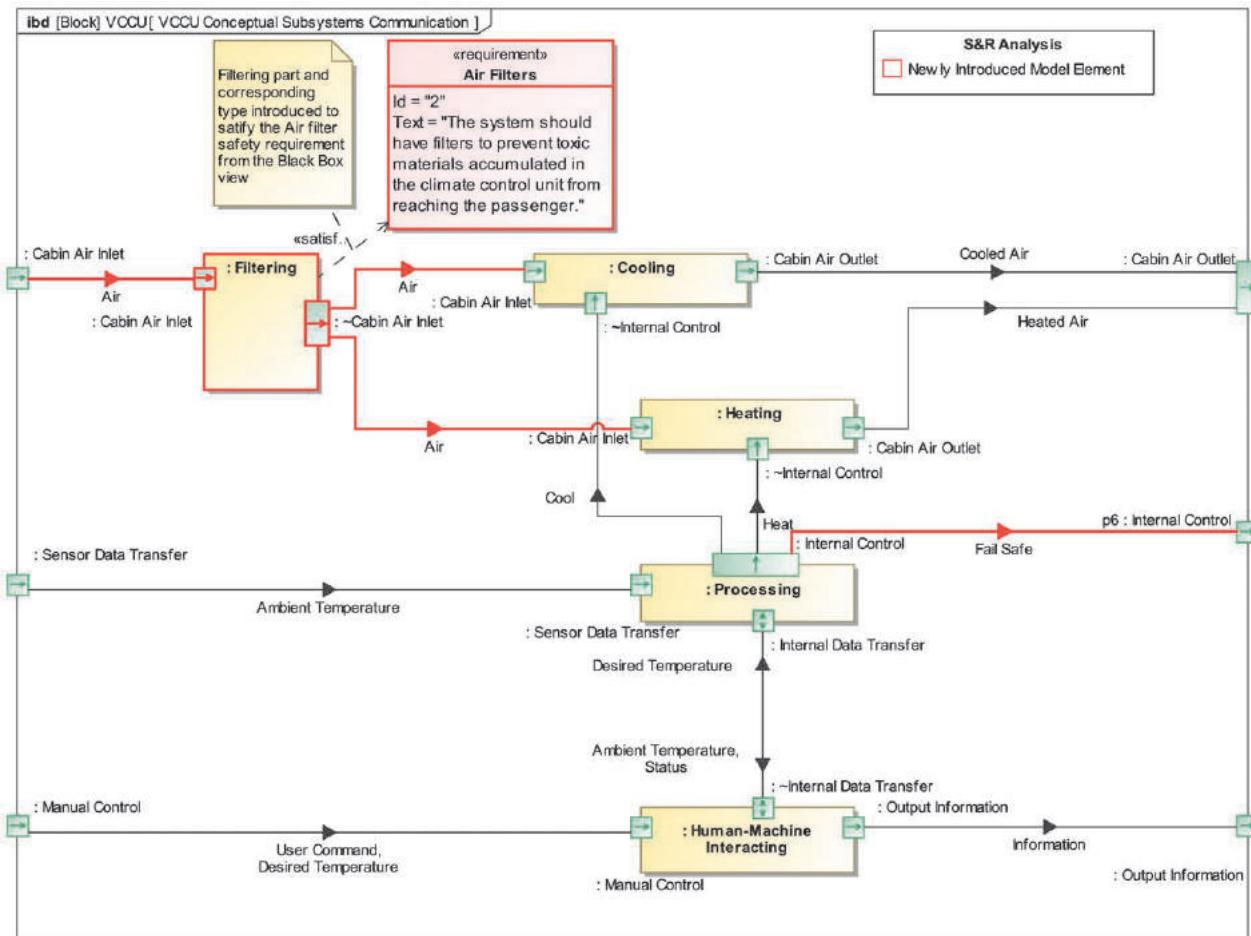
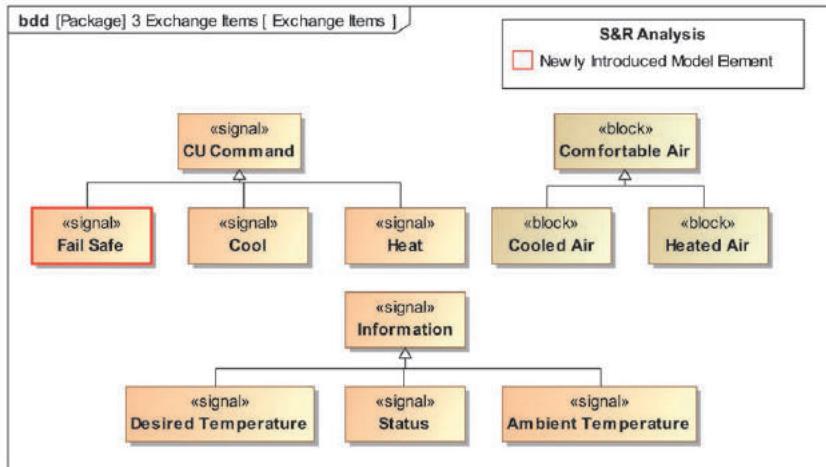


The Satisfy relationship is created from the Filter Air action and :Filtering subsystem to the 2 Air Filters safety requirement.

After making the aforementioned changes to the model, the *Biofouling Relation Map* is updated with the newly introduced model elements that satisfy the 2 Air Filters safety requirement.



Let's look at another example where the *5 Shut Down the System in Fail-Safe Manner when Unable to Read or Accept Ambient Temperature* safety requirement is addressed. The new action called *Fail Safe* is allocated to *:Processing* subsystem and the *Fail Safe* signal is emitted by the *Fail Safe* action to the outside of the Sol. The Fail Safe signal is one of the sub-types of *CU Command* signal that could flow through an existing interface, as shown in the following figure.



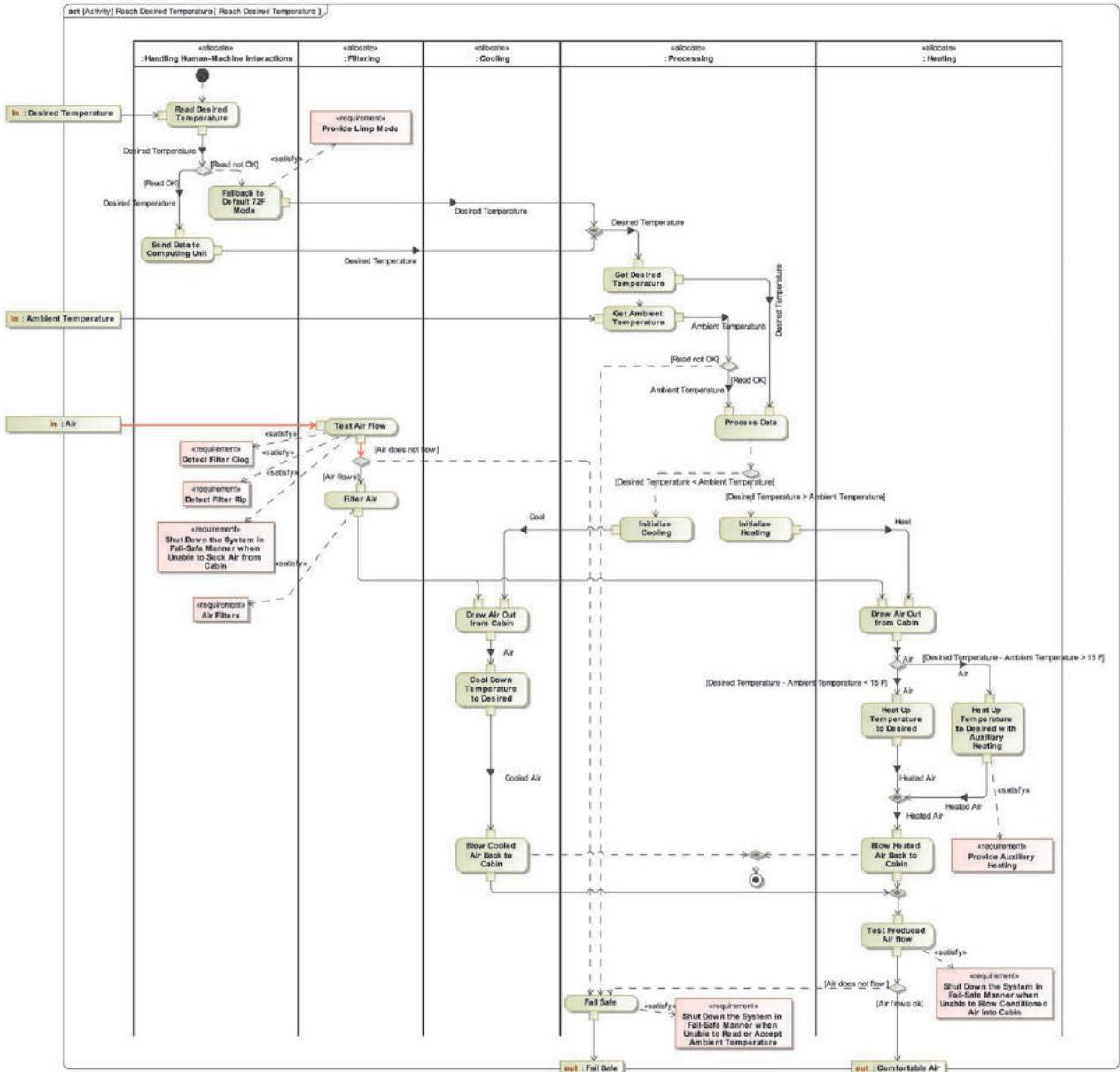
FMEA at the White Box view

Functional and conceptual FMEA needs to be performed at the White Box level just as it is done at the Black Box view: all functions, their incoming and outgoing flows, and conceptual subsystems need to be examined, as well as new conceptual subsystems and functions introduced to address safety and reliability requirements. For example, the *:Filtering* subsystem has to undergo the conceptual subsystems FMEA analysis (see the following figure). As a result of this analysis, new safety features need to be installed to address the safety of the *:Filtering* conceptual subsystem (since it can cause safety issues). For this, two new safety requirements *8* and *9* are created.

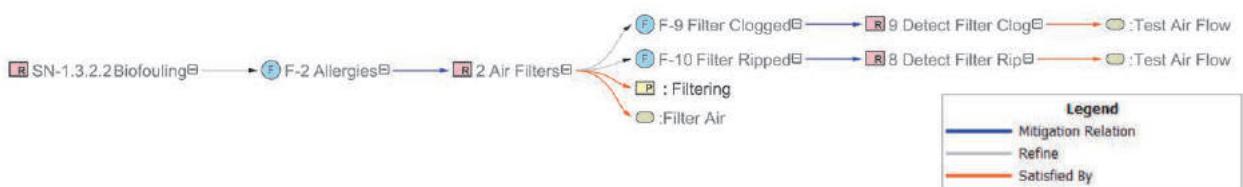
#	Id	Name	Item	Subsystem	Cause Of Failure	Failure Mode	Local Effect Of Failure	Final Effect Of Failure	Detection Control	Refines	Mitigation
1	F-9	Filter Clogged	VCCU : VCCU	: Filtering	Microfouling (dust, spores) Macrofouling (leaves, trash)	Reduction of function	VCCU Overloaded VCCU Overheated VCCU on Fire VCCU not being able to reach required temperature in time	Burns from fire Direct death from fire Poisoning from smoke Passangers overheated or undercooled Accident while driving	Detect Filter Clog	2 Air Filters F-1 VCCU on fire due to internal fault	9 Detect Filter Clog
2	F-10	Filter Ripped	VCCU : VCCU	: Filtering	Vibrations	Reduction of function	Direct contact of a passenger with toxic materials accumulated in climate control unit.	Discomfort while operating VCCU Allergic reactions affecting skin or pulmonary system	Detect Filter Rip	2 Air Filters F-2 Allergies	8 Detect Filter Rip

For example, the *F-9* FMEA item analyzes how the *:Filtering* subsystem can fail to protect the passenger from contact with toxic materials that may accumulate in the *:VCCU*, resulting in possible death from fire or poisoning from smoke due to a clog in the filter caused by external factors such as dust, leaves, or trash. As a result, the failure mode *Reduction of function* occurs, meaning the reduced ability of *:Filtering* subsystem to filter the air, resulting in local effects on the *:VCCU* (overloading or catching on fire), and final effects on the passenger, such as being burned, poisoned by smoke, or having an accident while driving, among others. To mitigate these risks a new safety requirement is created to detect any clog in the *:Filtering* subsystem.

The newly introduced safety requirements need to be addressed by making modifications to the systems model. The following figure depicts the *Test Air Flow* function that addresses these two requirements by testing incoming air flow. The diagram also contains the other safety requirements introduced at the Black Box level that are addressed and satisfied by appropriate functions.

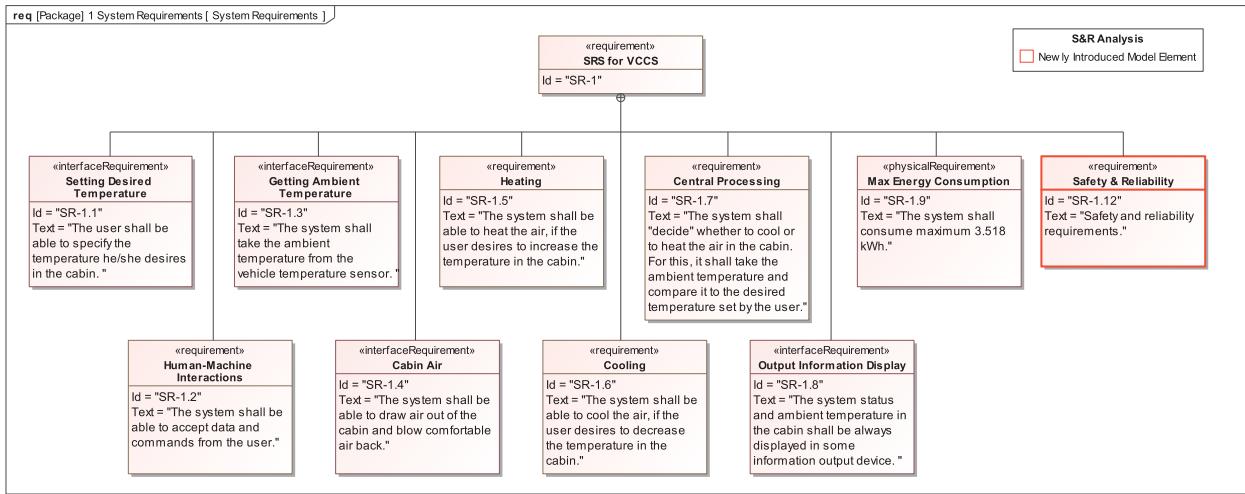


After the *:Filtering* subsystem examination, additional failures were found following fresh changes in the model. It all boils down to a sequence of possible failures that can affect lower-level system elements and model elements that mitigate the risks. The following figure illustrates the enriched cause-effect chain after the *:Filtering* subsystem conceptual subsystems FMEA analysis.



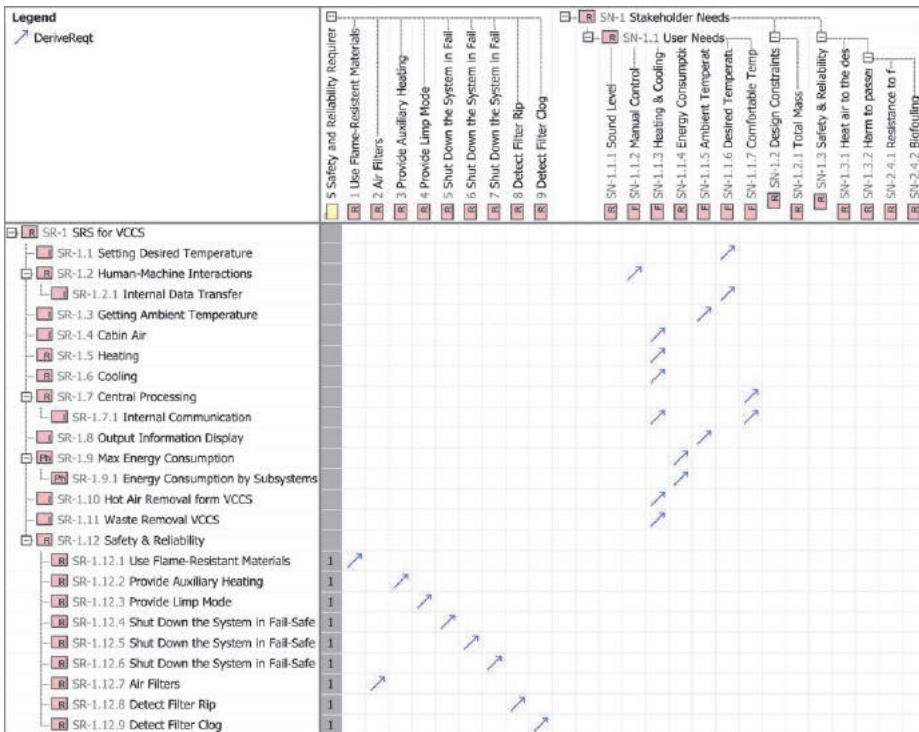
Addressing safety and reliability concerns in the LSA

At the solution level, safety and reliability requirements are derived from the problem domain. The diagram below shows a new *Safety and Reliability* grouping requirement that has safety and reliability requirements underneath.



#	△ Name	Text
1	□ R SR-1.12 Safety & Reliability	Safety and reliability requirements.
2	□ R SR-1.12.1 Use Flame-Resistant Materials	Use materials of HMIS flammability class I or less.
3	□ R SR-1.12.2 Provide Auxiliary Heating	Provide enough power to heat air to the desired temperature in 5 minutes.
4	□ R SR-1.12.3 Provide Limp Mode	VCCU shall be able to operate in limp mode by automatically keeping 72F temperature in the cabin.
5	□ R SR-1.12.4 Shut Down the System in Fail-Safe Manner w/o User Input	Shut down the system in fail-safe manner when unable to read or accept ambient temperature.
6	□ R SR-1.12.5 Shut Down the System in Fail-Safe Manner w/o User Input	Shut down the system in fail-safe manner when unable to suck air from cabin.
7	□ R SR-1.12.6 Shut Down the System in Fail-Safe Manner w/o User Input	Shut down the system in fail-safe manner when unable to blow conditioned air into cabin.
8	□ R SR-1.12.7 Air Filters	The system should have filters to prevent toxic materials accumulated in the climate control unit from reaching the passenger.
9	□ R SR-1.12.8 Detect Filter Rip	Detect Filter Rip.
10	□ R SR-1.12.9 Detect Filter Clog	Detect Filter Clog.

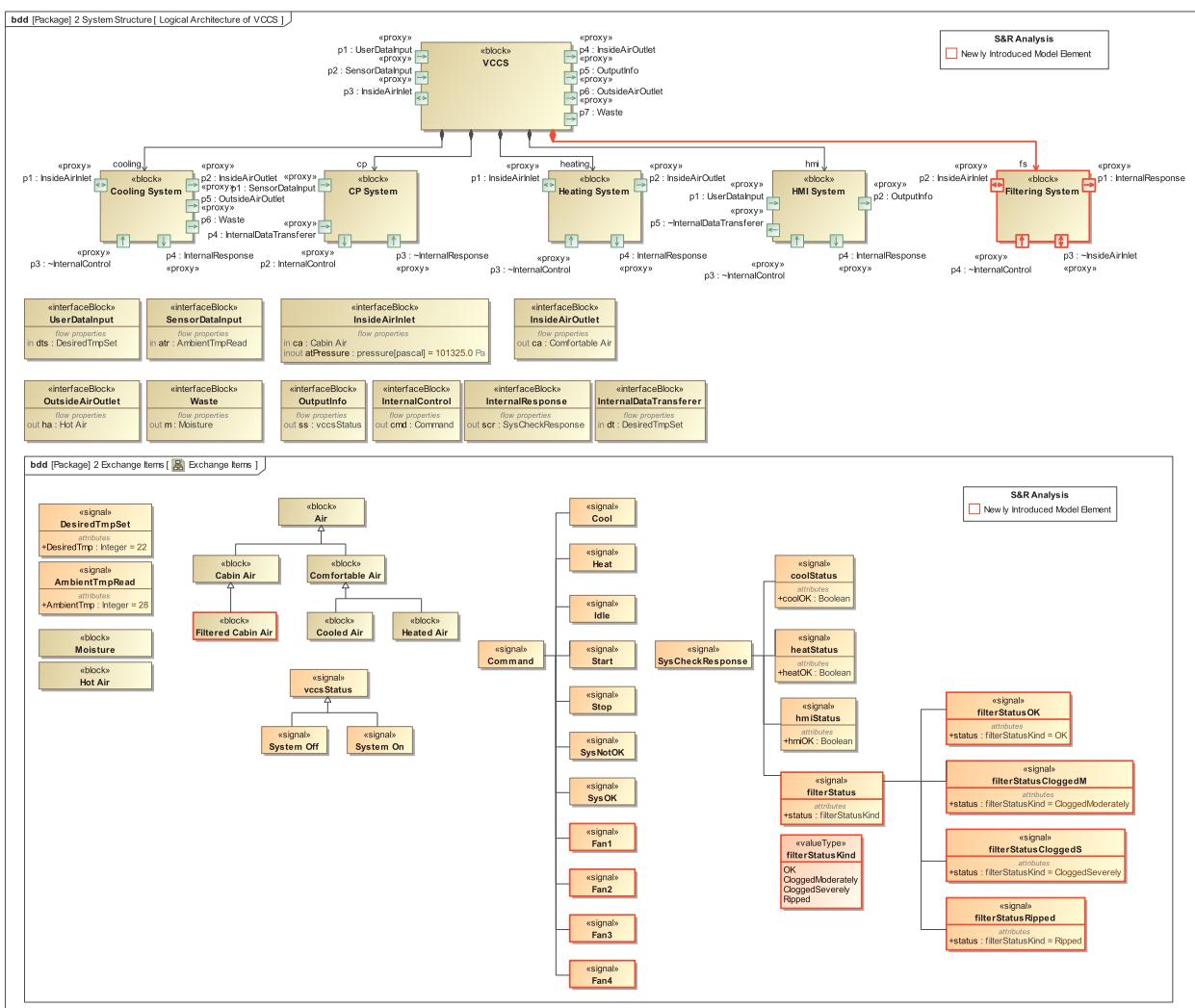
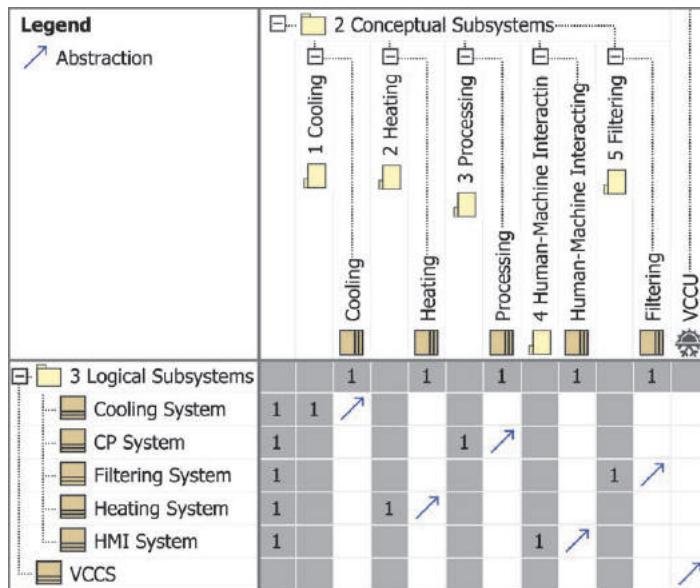
These safety and reliability requirements originate from the problem domain, as shown in a Derive Requirement Matrix called *System Requirements to Stakeholder needs*, as shown in the following figure.



These requirements, just like safety and reliability requirements in the problem domain, need to be addressed by making modifications to the model. For example, to address the *SR-1.12.7 Air Filters*

requirement, a *Filtering System* block needs to be created with corresponding ports and parts in the LSA level (see the following figures).

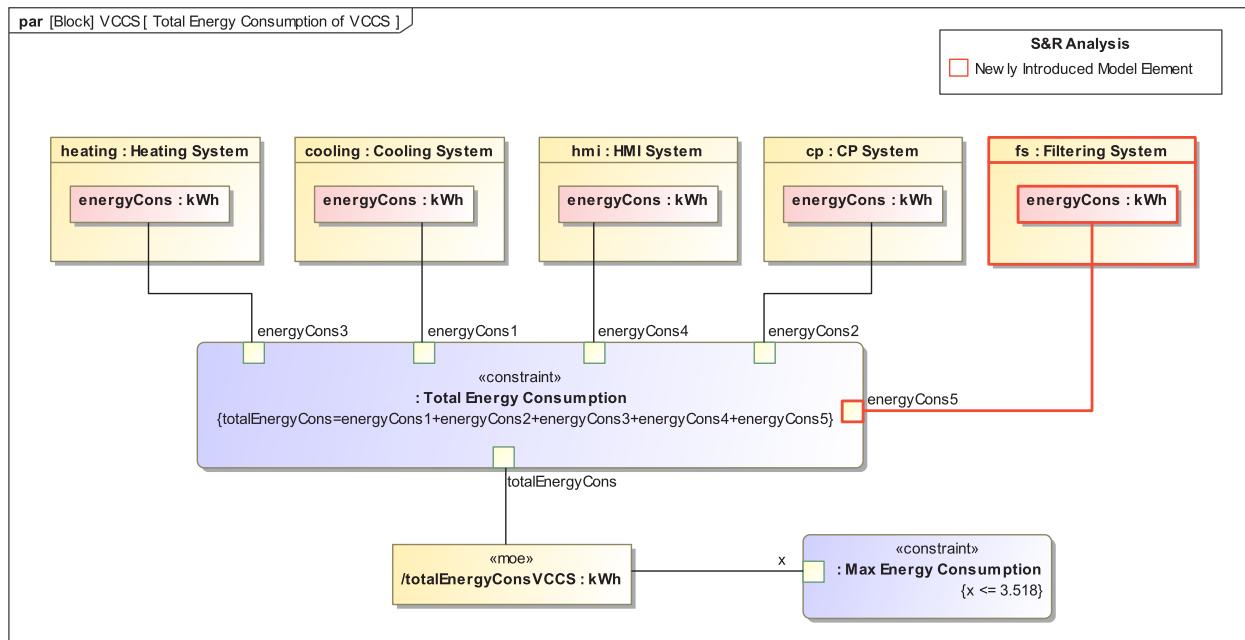
The «abstraction» relationship is used to declare that the *Filtering System* in the LSA is derived from the conceptual subsystem called *Filtering* in the problem domain analysis model. This relationship is illustrated in the *LSA to Problem Domain* matrix provided below.



The total energy consumption of the VCCS equals a sum of energies consumed by all its subsystems, meaning the total energy consumption expression has to be supplemented with the energy consumed by the Filtering System as well (see the [System Parameters](#) section). The constraint expression can be defined as follows:

totalEnergyCons = energyCons1 + energyCons2 + energyCons3 + energyCons4 + energyCons5

where *energyCons5* constraint parameter may stand for the energy consumed by the Filtering System. The *energyCons* value property needs to be created for the Filtering system and an additional *energyCons5* constraint parameter specified for the *Total Energy Consumption* constraint block. Once defined, the binding between the former value property and the constraint parameter needs to be established to enable the constraint expression to perform calculations on the modified model.



After these changes are complete it is time to specify the solution architecture of Filtering System and revisit other VCCS subsystems in order to ensure integrity at the system level.

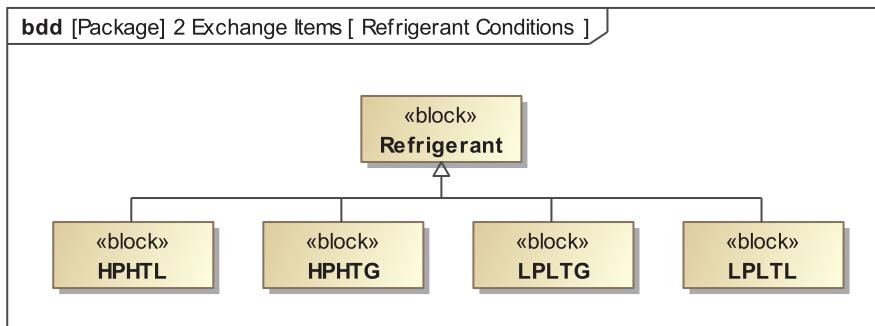
Addressing safety and reliability concerns at the subsystem level

Cooling Subsystem level

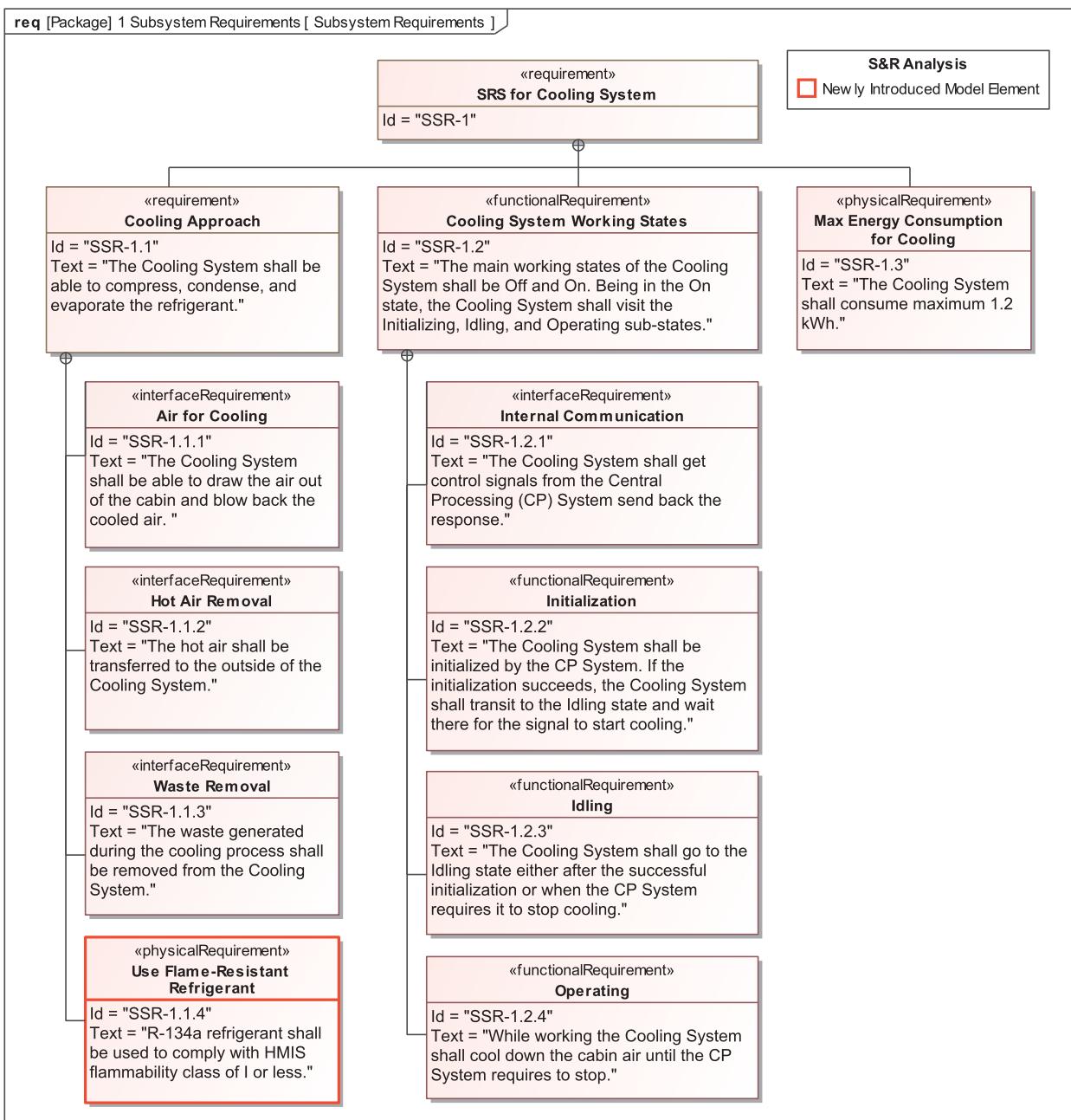
As mentioned in the Functional FMEA in the Black Box view section, the newly introduced requirements can be addressed either at the problem level or downstream. The *SR-1.12.1 Use Flame-Resistant Materials* safety requirement is one of the requirements that was introduced early at the problem domain level but could not be addressed due to a lack of specific materials in the design of the Sol. However, the *Cooling* subsystem architecture is detailed enough to be able to address this requirement.

The *Cooling* subsystem has already specified interactions for exchanging the *refrigerant*. The physical conditions of the refrigerant are already captured (see following figure). However, the *SR-1.12.1 Use Flame-*

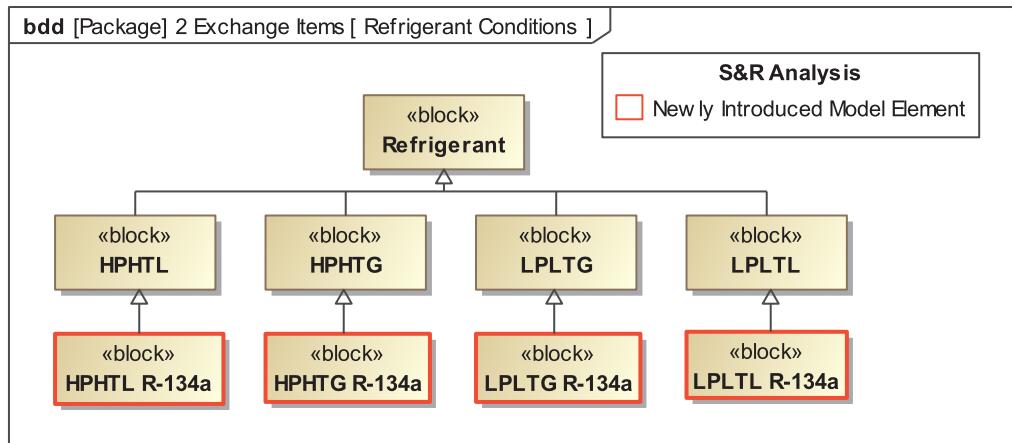
Resistant Materials safety requirement introduces the Hazardous Materials Identification System (HMIS) flammability class that the used material (in this case Refrigerant) shall comply with.



Additional *Cooling* Subsystem physical requirement called *SSR-1.1.4 Use Flame-Resistant Refrigerant* is introduced to address the *SR-1.12.1 Use Flame-Resistant Materials* safety requirement.



To satisfy this safety requirement the modification on the refrigerant type hierarchy needs to be performed. Therefore, the concrete *R-134a* refrigerant type can be defined as a block which can be captured as a sub-type of the each physical condition of the *Refrigerant*. The following figure contains the changes that were performed and illustrates how the *SSR-1.1.4 Use Flame-Resistant Materials* requirement is addressed.



Filtering Subsystem level

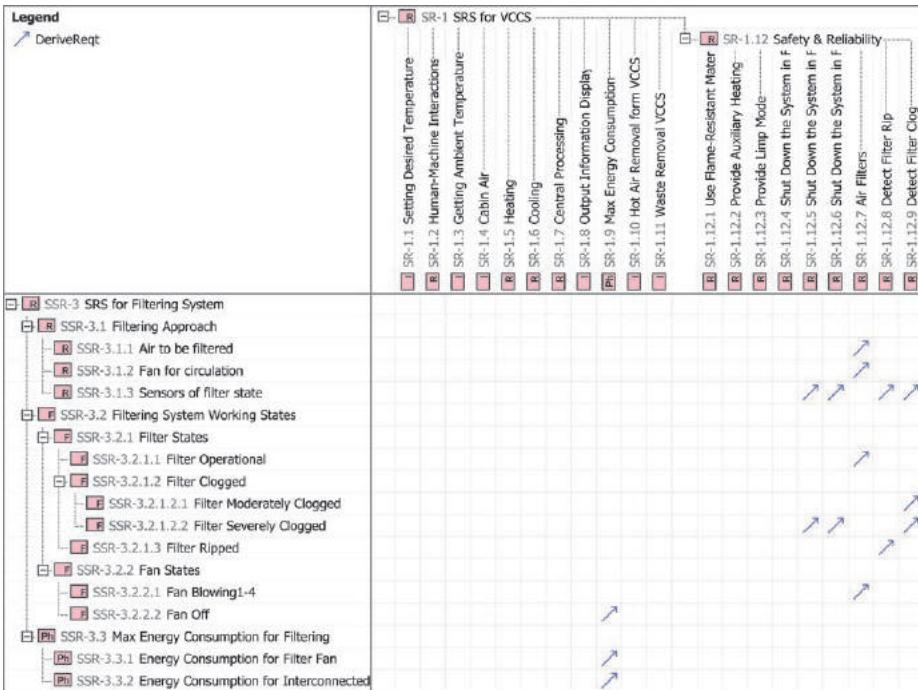
The engineering team has to begin working on logical subsystem architectures (LSSA) after the FMEA analysis has taken place and the new :*Filtering* subsystem has been identified (see Chapter [Building the logical architecture of subsystems](#)). As noted in the Building the logical architecture of subsystems section, an LSSA of each logical subsystem should be stored in a separate model. Filtering System can also have its own FMEA analysis.

Subsystem Requirements

Let's say that the engineering team working on the solution architecture of the Filtering System receives the following subsystem requirements from the System Architect:

#	Name	Text
1	SSR-3 SRS for Filtering System	
2	SSR-3.1 Filtering Approach	The simple mechanical filter with HEPA capability and additional fan to compensate for pressure losses.
3	SSR-3.1.1 Air to be filtered	The Filtering System shall be able to draw the air out of the cabin, filter it for particles and supply it to the downstream conditioning components (heater, cooler).
4	SSR-3.1.2 Fan for circulation	The fan shall be introduced to move the air around.
5	SSR-3.1.3 Sensors of filter state	Two pressure sensors to detect filter clogging and/or rip.
6	SSR-3.2 Filtering System Working States	Filter is always working unless in error condition of Clogged or Ripped. Fan can be in 4 levels of Blowing states or Off (in case filter takes air from outside of the vehicle and air flow is provided by inrush air pressure).
7	SSR-3.2.1 Filter States	
8	SSR-3.2.1.1 Filter Operational	Normal state of the filter. Filter removing particles of .3 um or larger with 95% efficiency.
9	SSR-3.2.1.2 Filter Clogged	
10	SSR-3.2.1.2.1 Filter Moderately Clog	Filter is moderately clogged. This causes (just) reduced functioning of the filter. Provide indicator for the user to visit the service.
11	SSR-3.2.1.2.2 Filter Severely Clog	Filter is severely clogged. This necessitates ventilator stop and disabling the HVAC function downstream. Provide indicator for the user to visit the service.
12	SSR-3.2.1.3 Filter Ripped	Filter is ripped. This causes reduced functioning of the filter. Provide indicator for the user to visit the service for repairs/replacement.
13	SSR-3.2.2 Fan States	
14	SSR-3.2.2.1 Fan Blowing1-4	Fan can be running in one of the 4 power states from Blowing1 (least powerful) to Blowing4 (max power).
15	SSR-3.2.2.2 Fan Off	Fan can be off.
16	SSR-3.3 Max Energy Consumption for Filter	The Filtering System shall consume maximum 300 W.
17	SSR-3.3.1 Energy Consumption for Filter	Filtering system shall not use more than 150W for the fan power. Power is dependent on fan level according to formula: BlowingLevel x 37.5W
18	SSR-3.3.2 Energy Consumption for Intake	Filtering system shall not introduce more than 150W of additional power load on connected components downstream (heater, cooler).

Matrices and maps can be used to establish and review the cross-cutting relationships between the :*Filtering* subsystem requirements and system requirements. The matrix below captures which more concrete :*Filtering* subsystem requirements were derived from what system requirements.



A Relation map can be created, where blue arrows represent the relationships between requirements from different domains, red arrows represent the relationships between failures and requirements that mitigate the failures, and cyan arrows represent the refinement relationships between failures that were created to refine requirements from different domains. All these relationships illustrate the FMEA cause-effect chains from initial stakeholder requirements down to Filtering System requirements in accordance with safety and reliability decisions made during FMEA analysis. The following figure shows a portion of the relation map called from the *SN-1.3.2.2 Biofouling* stakeholder requirement down to the Filtering System.



Subsystem Structure

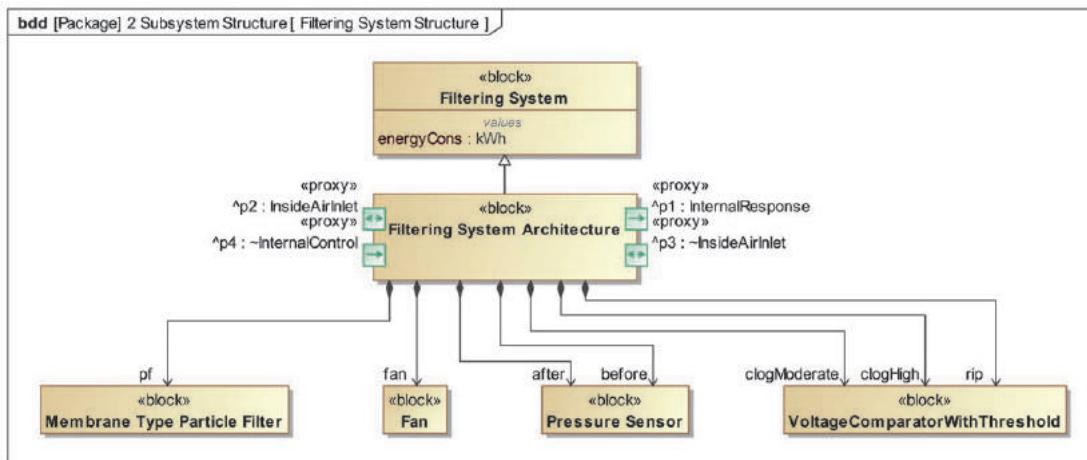
The internal structure of the Filtering System can be identified, then operations among its components, and how logical components of the Filtering System interact with other subsystems of the same **Sol** can be specified. To find the detailed steps refer to Chapter [Subsystem Structure](#).

Capturing components of the Filtering system

As defined by the subsystem requirements and based on the engineering team experience, the Filtering system shall be composed of:

- Fan
- Membrane Type Particle Filter
- Pressure Sensor
- Voltage Comparator With Threshold

These can be specified as part properties of the Filtering System block. For this, the SysML block definition diagram (bdd) is used, as shown in the following figure.

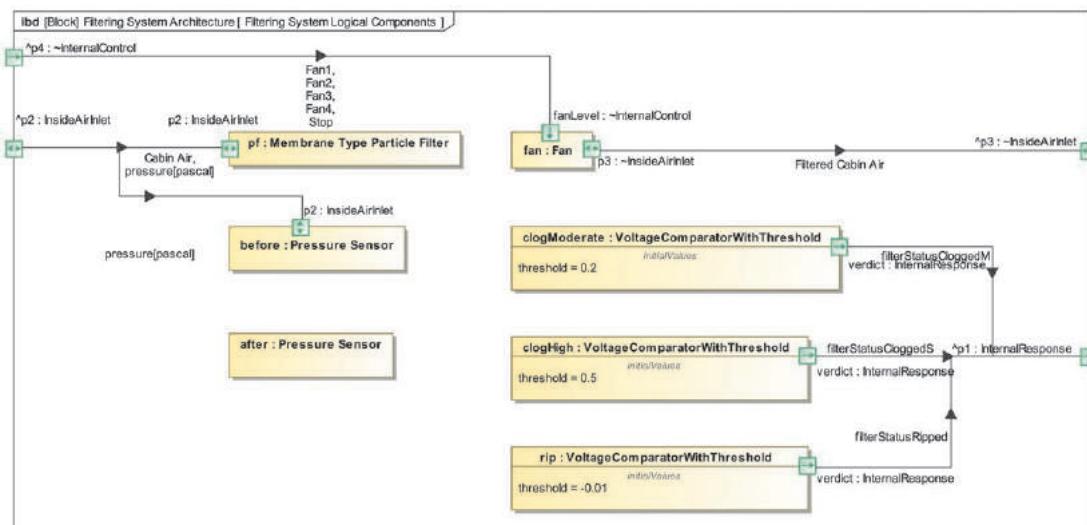


It should be noted that there is a need for the system to be constantly aware of filter status according to LSA SSR 3.2.1 *Filter Status* requirements group. To be able to operate securely the Filtering System shall monitor if the filter is ripped, clogged, and the severity of the clogging. Therefore, the pressure sensors are installed before and after the air has blown through the filter. Then the pressure information is compared by three dedicated voltage comparators to determine the status of the filter.

Specifying interactions within the Filtering System

In this step, we focus on specifying interactions between components of the Filtering System and outside of the subsystem. To specify interactions between subsystem components the SysML internal block diagram (ibd) is used as shown in the following figure. An ibd is created for the *Filtering System Architecture* block. As the procedure in [step 5](#) of the [Subsystem Structure](#) tutorial suggests, first, we start specifying interactions via inherited ports:

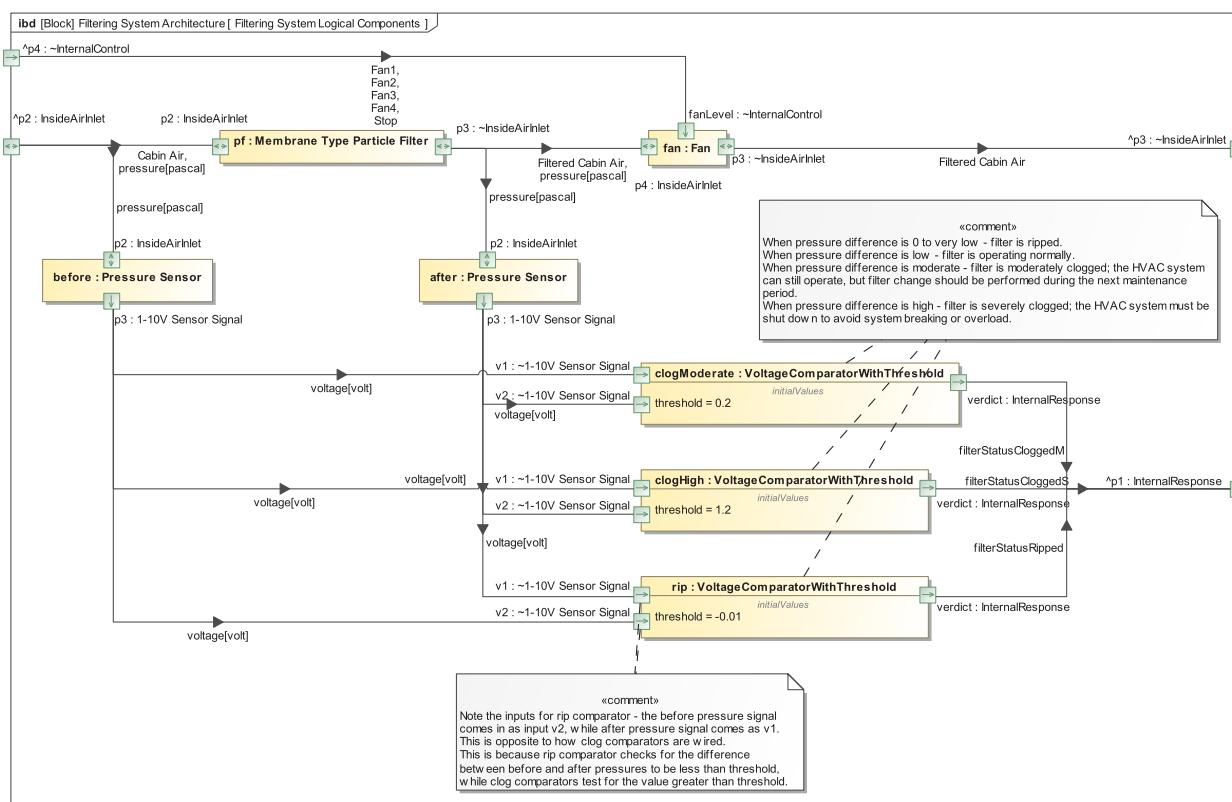
1. One for letting the cabin air and its pressure flow into the Filtering System that is consumed by : *Membrane Type Particle Filter* and *before : Pressure Sensor* components.
2. Second for letting the filtered cabin air out from the : *Fan* of the Filtering System.
3. Another dedicated to control the : *Fan* by passing over the signals: Fan1, Fan2, Fan3, Fan4, and Stop.
4. The last one is dedicated to emit the signals indicating the state of the Filtering System from voltage comparators out of the subsystem.



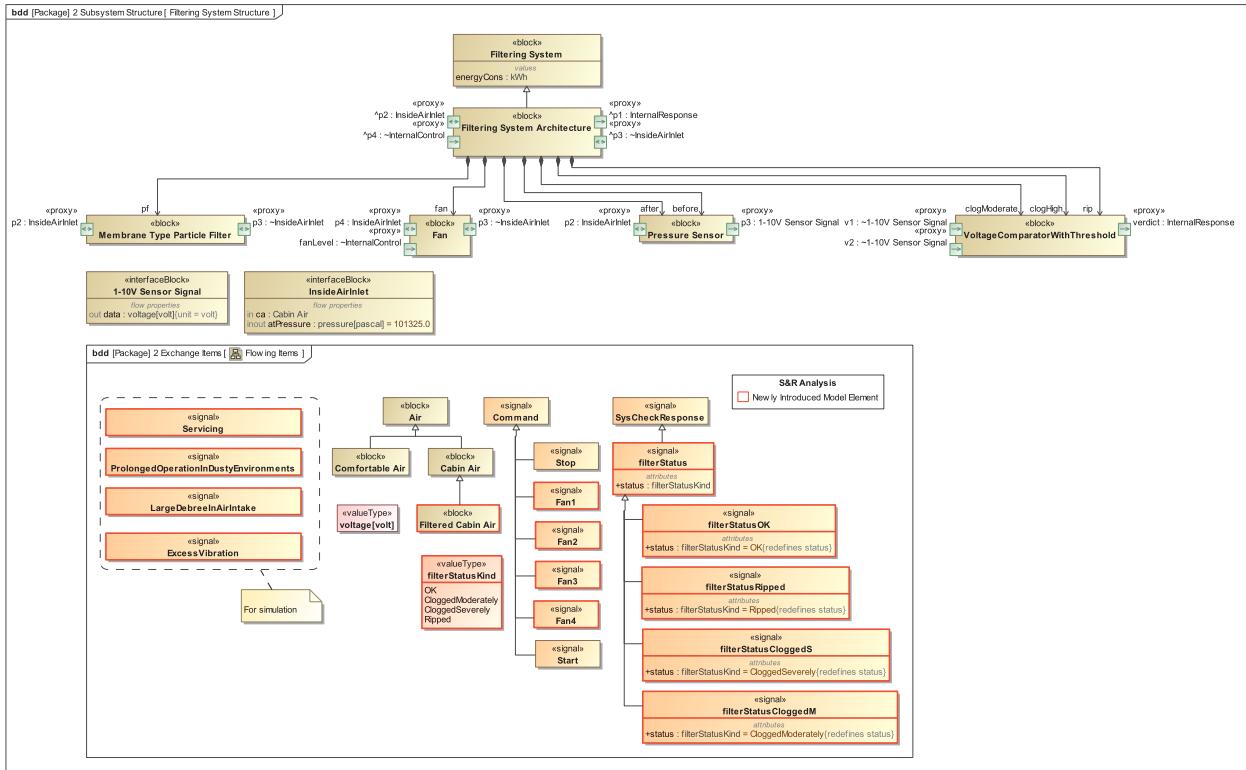
After interactions via inherited ports are captured the next step is to proceed with interactions among the Filtering System logical components. For more details refer to the [step 6](#) of the [Subsystem Structure](#) tutorial.

Connectors are created between compatible ports to convey the following interactions:

1. The *before : Pressure Sensor* converts air pressure to an electrical signal (voltage) proportional to the air pressure before filtering.
2. The *: Membrane Type Particle Filter* provides filtered air to *: Fan* and filtered air pressure to *after : Pressure Sensor*.
3. The *after : Pressure Sensor* consumes filtered air pressure and produces the voltage signal proportional to the air pressure after filtering.
4. The filtered air is consumed by *: Fan* in order to be circulated out of the system.
5. Pressure sensors produce the voltage signals to *clogModerate : VoltageComparatorWithThreshold*, *clogHigh : VoltageComparatorWithThreshold*, and *rip : VoltageComparatorWithThreshold* components where voltage signals of air are compared with the present voltage reference, called *threshold* before and after the filtering.



After identifying interactions between the components and the outside of the Sol, the next step is to update the *Filtering System Structure* bdd diagram to display new proxy ports on the blocks that capture the Filtering System Architecture and its components (see the following figure).



Subsystem Behavior

After identifying and capturing the structural architecture of the Filtering system, the next step is to continue with behavioral architecture modeling. The SysML state machine and activity or sequence diagrams are used to capture the behavior of the Filtering System.

Capturing states of the Filtering System

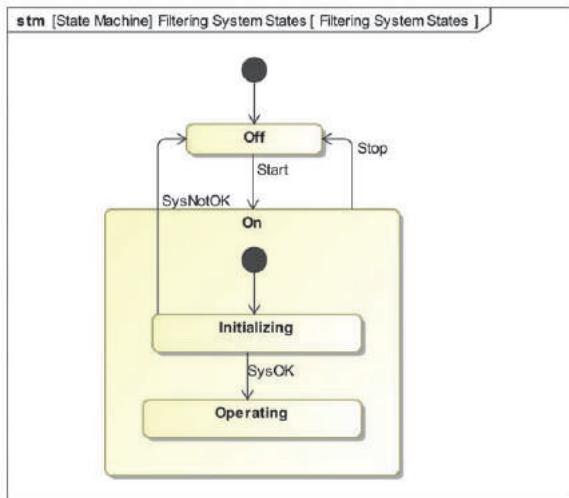
To capture the main states of the Filtering System the SysML state machine diagram is used, displayed in the following figure analogously as the Cooling subsystem. The transitions between states are triggered by a signal event, such as *Start* between the *Off* and *On* states or *SysNotOK* and *SysOK* signals between *Initializing*, *Off* and *Operating* states.

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

1. Off
2. On:
 - a. Initializing
 - b. Operating

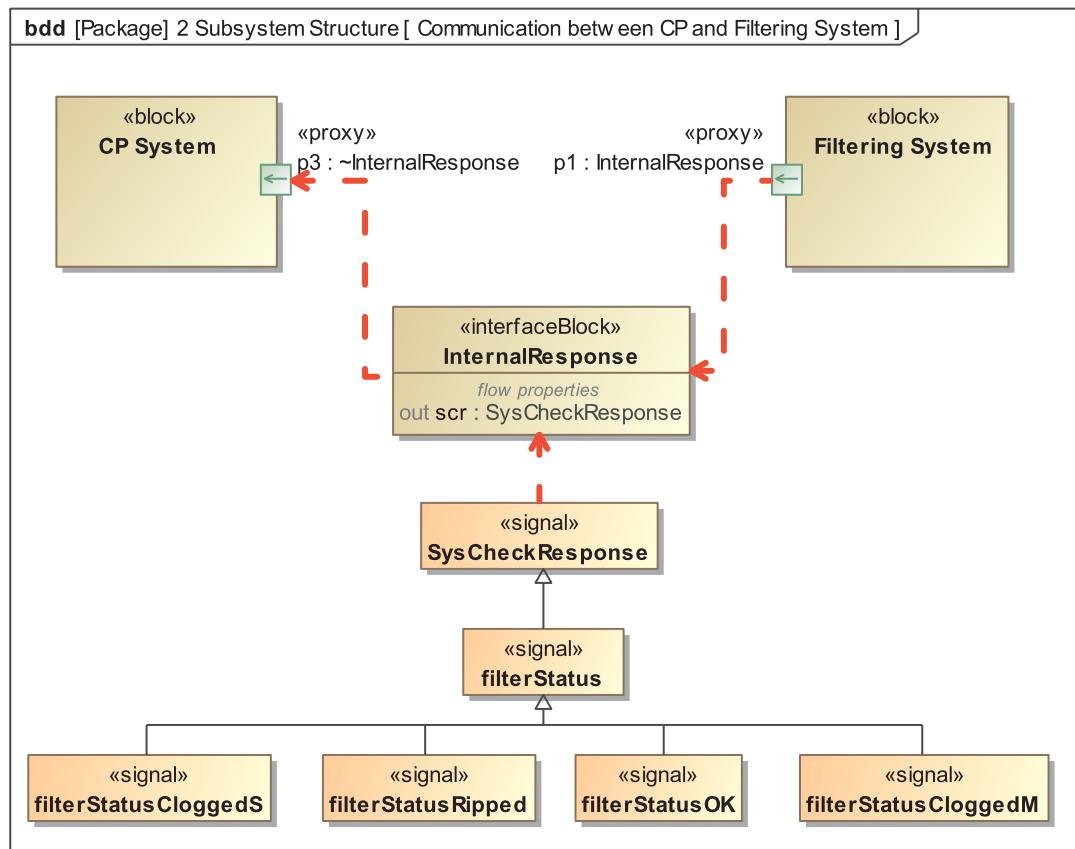
The following rules describe transitions between those states:

- The first state of the subsystem is *Off*.
- The subsystem can move from the *Off* state to the *On* state and back.
- When 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 *Off* state.



As in the case of the Cooling System, these signals come from outside the Filtering System as well. The CP System sends these signals to invoke the system. Thus, the reaction of that subsystem to the *Start* signal has to be specified. The Filtering System in response sends the *filterStatus* signal with the *status* attribute value set to *filterOK* if it is ready to start, and one of the values of *filterStatusCloggedM*, *filterStatusCloggedS* or *filterRipped*, if it is not. The following figure depicts the response signal that is sent over the *p1* proxy port.

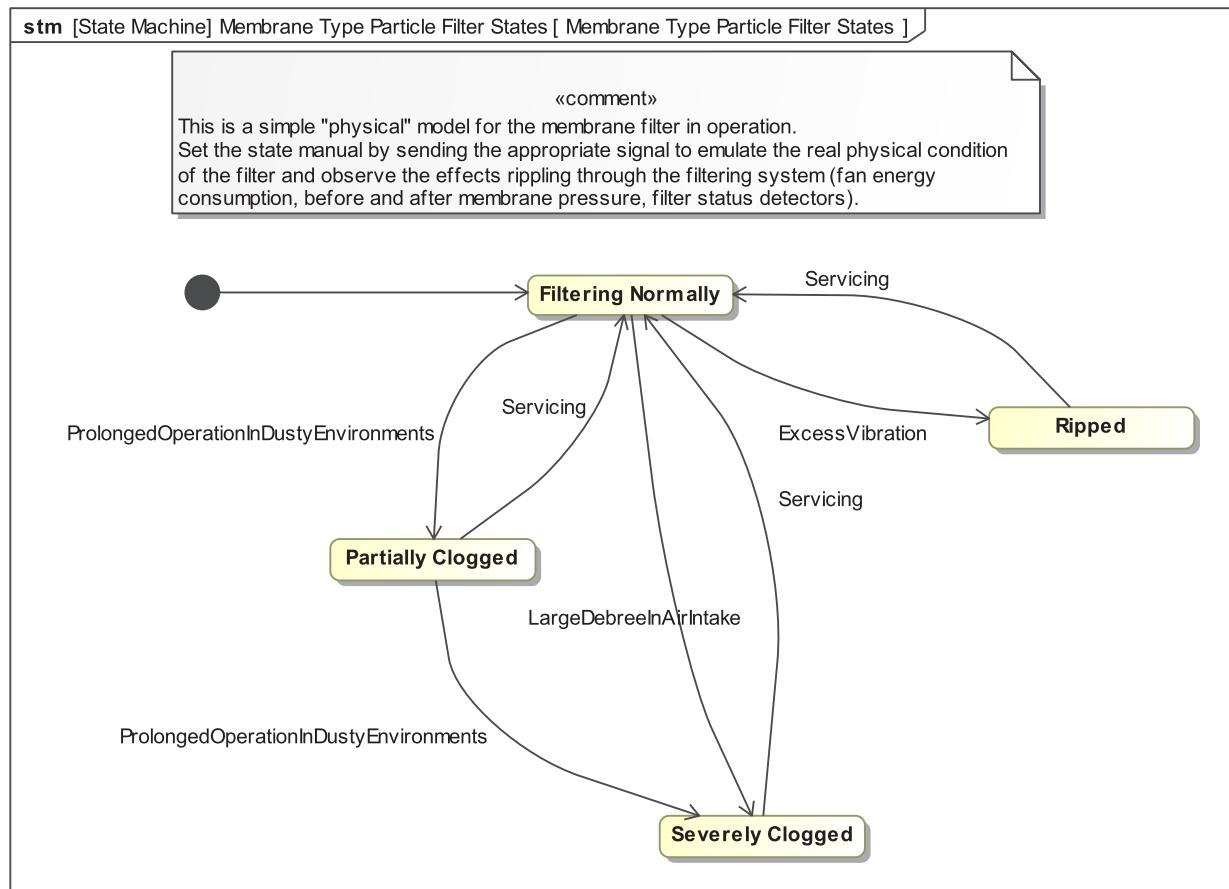
i *filterStatusCloggedM* stands for filter Status Clogged Moderately, and *filterStatusCloggedS* stands for filter Status Clogged Severely.



The Membrane Particle Filter is the concrete component that can physically be damaged and cause the Filtering system failure to initiate. Therefore, the Membrane Particle Filter can have its own state machine indicating the states in which it can exist, such as:

1. Filtering Normally
2. Partially Clogged
3. Severely Clogged
4. Ripped

These can be triggered by such environmental conditions as *ProlongedOperationInDustyEnvironments*, *LargeDebreelInAirIntake* or *ExcessVibration* that are depicted as signal events on the transitions in the Membrane Type Particle Filter States diagram. When the Membrane Particle defect is removed it can resume working normally. For this, the *Servicing* signal is created to indicate that component was repaired.



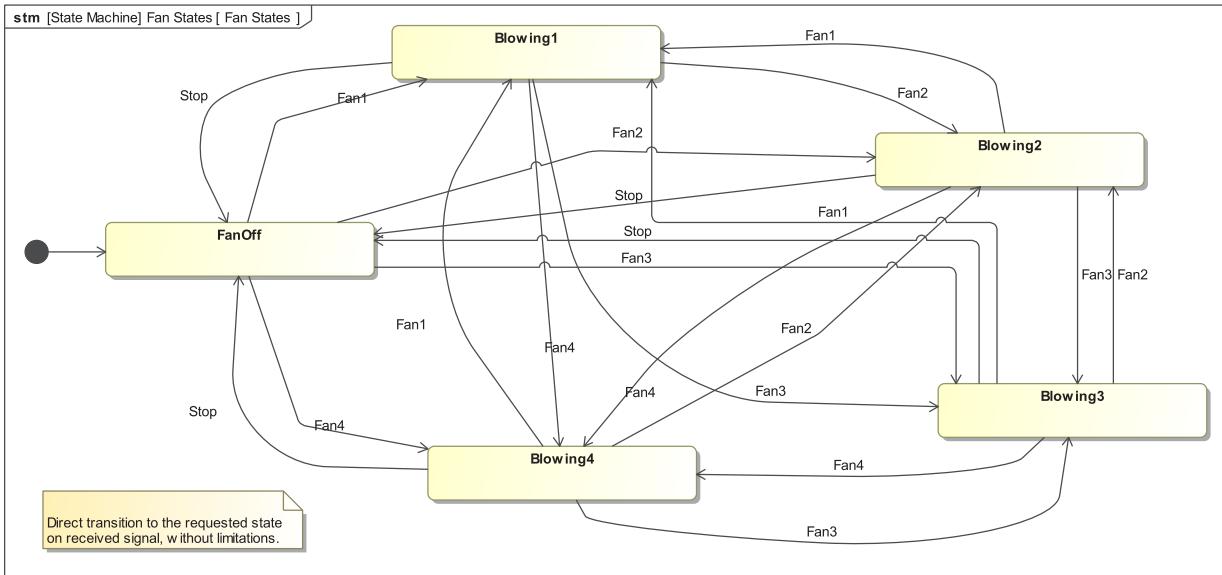
To address requirements under the *SSR-3.2.2 Fan States* grouping requirement, another state machine is created for Fan component of the Filtering System (see the following figure). The Fan State diagram depicts the power states in which Fan can exist during system operation, such as:

1. Off
2. Blowing1
3. Blowing2
4. Blowing3
5. Blowing4

These states are triggered by signal events *Fan1*, *Fan2*, *Fan3*, and *Fan4* to change the power of the Fan. The Fan also can be turned off any time despite its Blowing state. The *Stop* signal is used for this transition

to occur. The Fan is controlled by the CP System; thus, all triggering signals are being sent from the CP System.

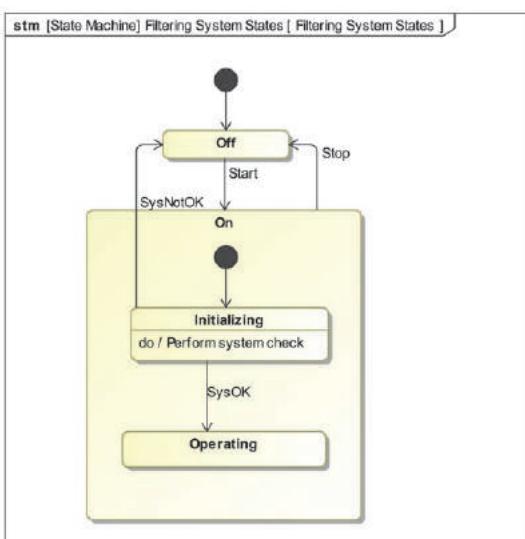
- i** The number along the Blowing state name represents the power of the fan, the bigger number means a more powerful mode of the fan. Signal names that trigger these states are named analogously for each of the states.



Specifying internal behaviors of the state

As mentioned in the **step 5** of the [Subsystem Behavior](#) tutorial, every logical subsystem of the **Sol** can perform one or more behaviors related to one or more of its states. The internal behavior of the Filtering System is captured using the SysML activity diagram. To indicate that the Filtering System performs a self-check and sends out the *filterStatus* signal with an appropriate attribute value, an activity diagram is created that is specified as a *do* behavior on the *Initializing* state.

To specify that the CP System evaluates the status of all subsystems and sends the *SysOK* and *SysNotOK* signals the CP System Evaluate Status state machine diagram should be updated. The following figure captures the final view of the *Filtering System States* state machine diagram.



The change has to be reconciled with other engineering teams to ensure integrity at the system level.

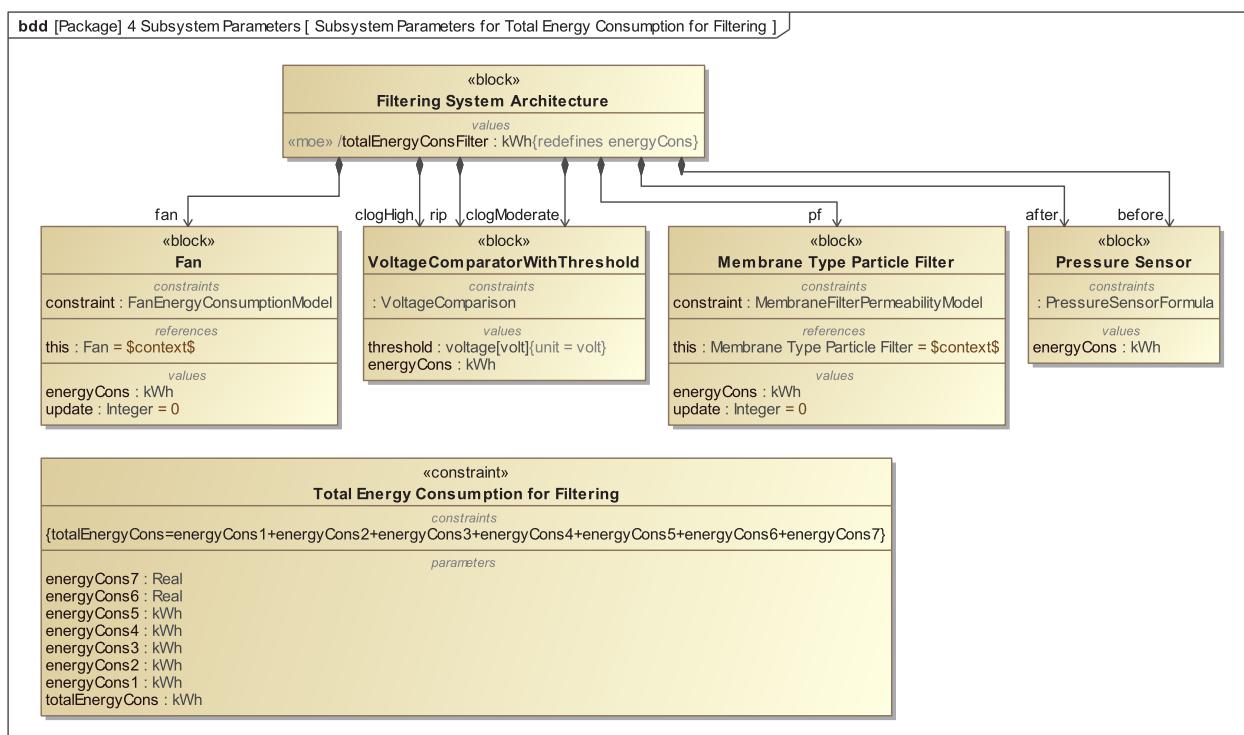
Subsystem Parameters

Subsystem parameters are used to capture quantitative characteristics of the Filtering logical subsystem that can be measured. They are used to calculate MoEs for that logical subsystem to satisfy non-functional quantitative subsystem requirements. Using the capabilities of the **modeling tool**, the model can be executed to calculate MoEs using mathematical expressions to determine whether quantitative requirements are satisfied or not. (See the [Subsystem Parameters](#) section.)

Specifying subsystem parameters for calculating the total energy consumption for filtering

According to the subsystem requirements specification and the MoEs identified for the Filtering System captured in the problem domain and downstream, the formula for calculating the total energy consumed by the Filtering System needs to be modeled. The SysML block definition diagram (bdd), called *System Parameters for Total Energy Consumption for Filtering*, is used to specify constraint expressions and parameters necessary to define *totalEnergyConsFilter* value property.

The *Total Energy Consumption for Filtering* constraint block depicted in the following figure has seven input parameters, each for the energy consumption of a single component of the Filtering System, and therefore requires seven component parameters to be defined in the model. Value properties should be specified for each block that represents components of the Filtering System.



Let's say the total energy consumption of the Filtering System equals the sum of energies consumed by all its components. The constraint expression can be defined as follows:

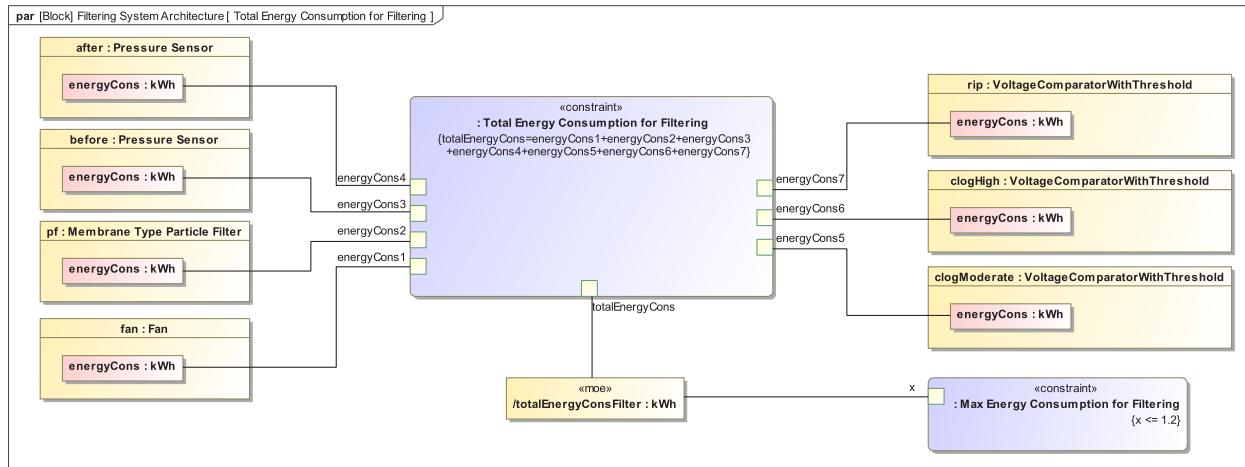
$$totalEnergyCons = energyCons1 + energyCons2 + energyCons3 + energyCons4 + energyCons5 + energyCons6 + energyCons7$$

where the following constraint parameters:

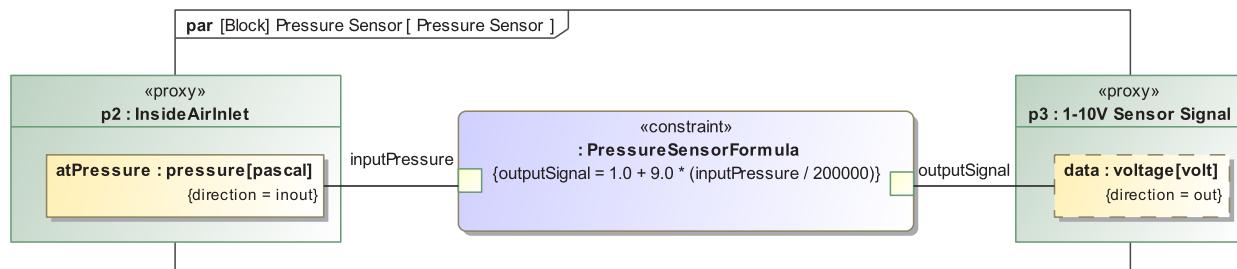
1. *totalEnergyCons* stands for the total energy consumed by the Filtering System
2. *energyCons1* may stand for the energy consumed by the Fan
3. *energyCons2* may stand for the energy consumed by the Membrane Type Particle Filter
4. *energyCons3* may stand for the energy consumed by the before : Pressure Sensor

5. *energyCons4* may stand for the energy consumed by the after : Pressure Sensor
6. *energyCons5* may stand for the energy consumed by the rip : VoltageComparatorWithThreshold
7. *energyCons6* may stand for the energy consumed by the clogHigh : VoltageComparatorWithThreshold
8. *energyCons7* may stand for the energy consumed by the clogModerate : VoltageComparatorWithThreshold

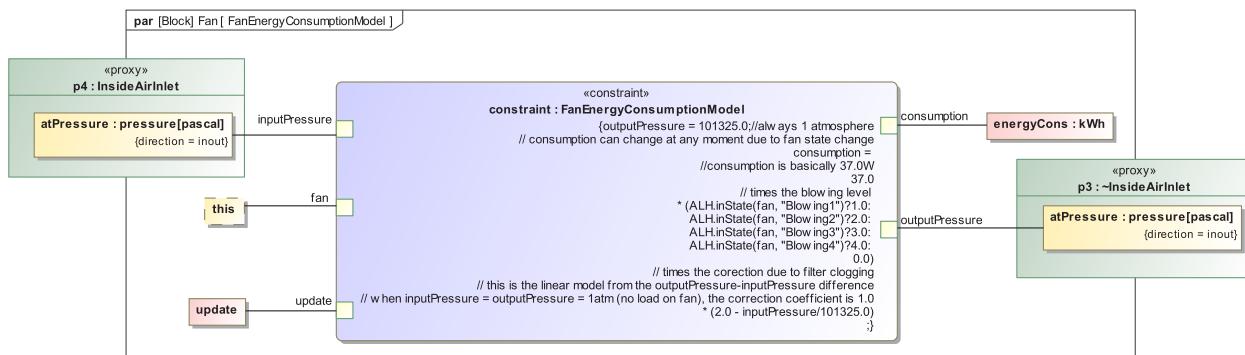
should be bound to corresponding value properties specified in the model. For this, the SysML parametric diagram called *Total Energy Consumption for Filtering* is used, which is illustrated in the following figure. The *Max Energy Consumption for Filtering* subsystem requirement requires that “*The Filtering System shall consume maximum 1.2 kWh*”. To satisfy this requirement the constraint block is created and bound with the *totalEnergyConsFilter* value.



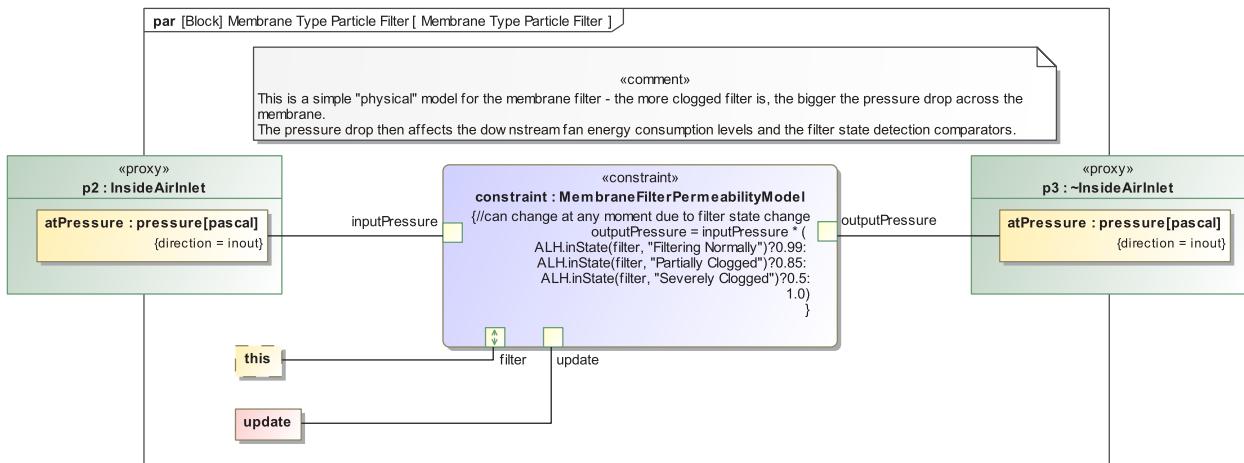
This is a simple sensor model, converting input pressure in the range 0-2 atmospheres (0-200000 Pa) into the voltage output signal in 1-10V range, that is depicted using parametric diagrams presented in the following figure.



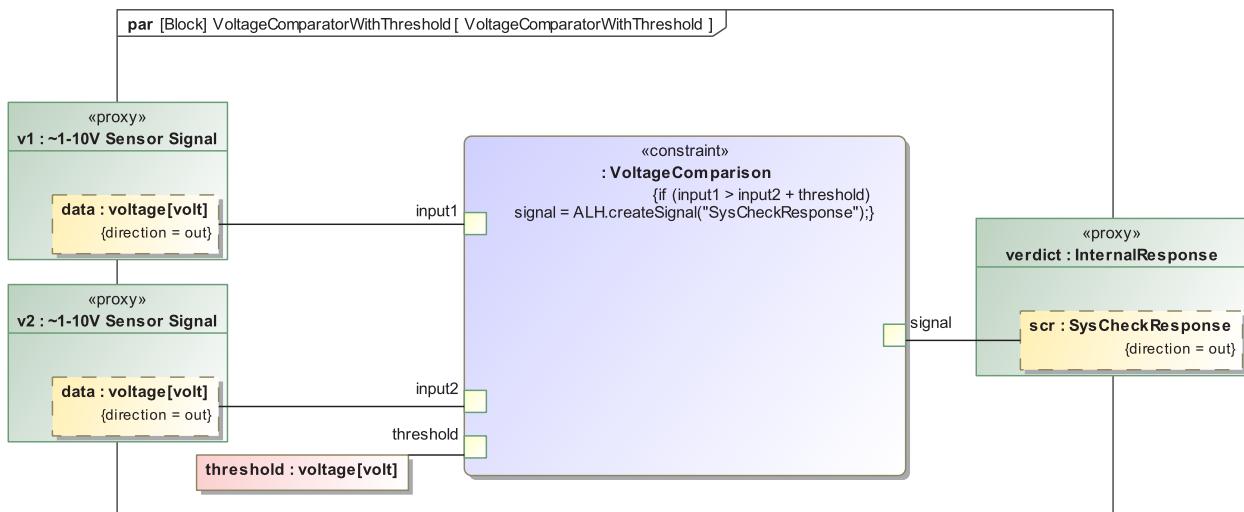
The second figure illustrates the simplified model for fan energy consumption depending on the : *Fan* state and the input pressure, which in turn depends on the pressure drop in the membrane filter permeability). The consumption is 37 Watts times the : *Fan* blowing intensity (Blowing1-Blowing4) times the linear correction factor depending on the input pressure drop.



This is a simple membrane filter model, which models pressure drop through the filtering membrane depending on the filter state (clogged/ripped/normal). The pressure drop then affects downstream systems, such as the energy consumption of the :Fan.



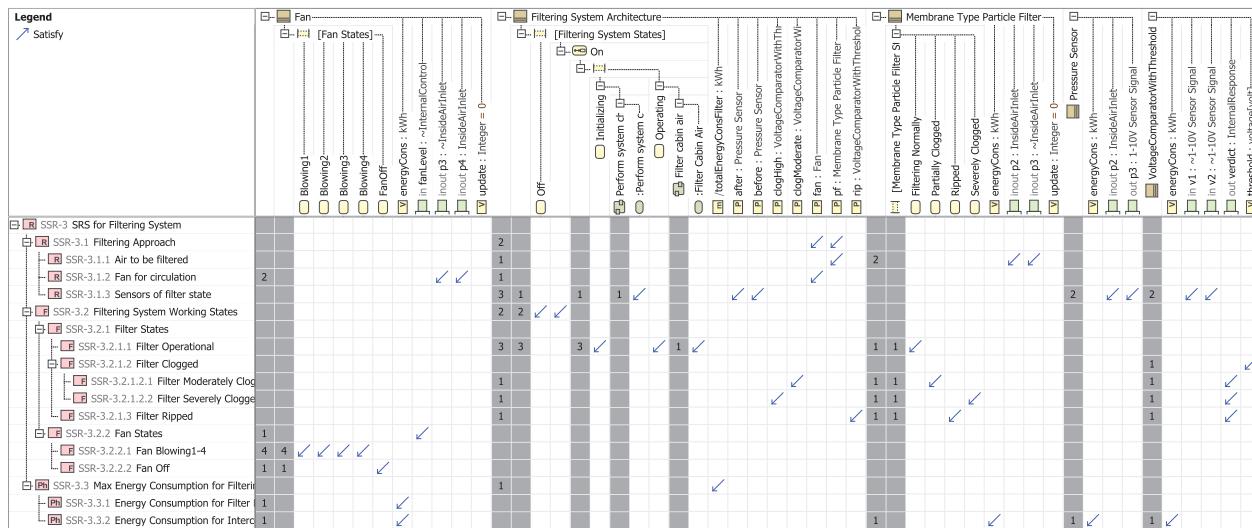
This is a model for a voltage comparator with a threshold. When the voltage at input1 is greater than input2 + threshold, an output signal is fired.



The next step is to establish traceability relationships from the elements that capture the LSSA of Filtering System to subsystem requirements.

Traceability to Subsystem Requirements

According to the [Traceability to Subsystem Requirements](#) section the cross-cutting relationships (such as satisfy) need to be defined in order for subsystem requirements to be fulfilled. As you can see in the following figure, the satisfy requirement matrix called *LSSA to Subsystem Requirements* is completely fulfilled, meaning that all subsystem requirements are satisfied by one or more architecture elements.



Logical subsystems FMEA at the subsystem level

As mentioned in the conceptual FMEA in the Black-Box view section, the safety and reliability analysis workflow is repetitive, thus safety and reliability analysis can also be performed for logical components of the VCCS. The following figure represents logical subsystems FMEA and safety and reliability analysis performed for the *FilteringSystem* with focus on the *before : Pressure Sensor* and *after : Pressure Sensor* components. The pressure sensors have to undergo safety and reliability analysis since the Filtering System may no longer detect malfunctions in case of the failure, and, as a consequence, the subsystem would no longer be able to prevent safety issues in the system.

#	△ Id	Name	Classification	Item	Subsystem	Cause Of Failure	Failure Mode	Local Effect Of Failure	Final Effect Of Failure	Detection Control	Mitigation
1	F-11	Upstream Pressure Sensor Failure	electrical	Filtering System Architecture	before : Pressure Sensor	Wire Breakage	Loss of function	Filter Clog State Unknown VCCU Overloaded VCCU Overheated VCCU on Fire	Poisoning from smoke Burns from fire Accident while driving	Detect Filter State Unknown	SSR-4 Handle Unknown Filter State
2	F-12	Downstream Pressure Sensor Failure	electrical	Filtering System Architecture	after : Pressure Sensor	Wire Breakage	Loss of function	Filter Clog State Unknown VCCU Overloaded VCCU Overheated VCCU on Fire	Poisoning from smoke Burns from fire Accident while driving	Detect Filter State Unknown	SSR-4 Handle Unknown Filter State
3	F-13	Upstream Pressure Sensor Failure clone	mechanical	Filtering System Architecture	before : Pressure Sensor	Sensing Orifice Clog	Loss of function	Filter Clog State Unknown VCCU Overloaded VCCU Overheated VCCU on Fire	Poisoning from smoke Burns from fire Accident while driving	Detect Filter State Unknown	SSR-4 Handle Unknown Filter State
4	F-14	Downstream1 Pressure Sensor Failure clone	mechanical	Filtering System Architecture	after : Pressure Sensor	Sensing Orifice Clog	Loss of function	Filter Clog State Unknown VCCU Overloaded VCCU Overheated VCCU on Fire	Poisoning from smoke Burns from fire Accident while driving	Detect Filter State Unknown	SSR-4 Handle Unknown Filter State

For example, the F-11 FMEA item analyzes how *before : Pressure Sensor* can fail to protect the passenger from accidents, such as burns from fire or poisoning due to adverse conditions formed in the VCCU, such as a clogged filter or VCCU on fire, that are called local effects of failure. These conditions can be caused by wire breakage in the *before : Pressure Sensor*, which would result in an inability to provide data about the air pressure to other subsystem parts. As a result, the failure mode *Loss of Function* occurs, meaning the lost ability to determine the state of the *pf : Membrane Particle Filter*. To mitigate the risk a new safety requirement is introduced to handle unknown filter status (see the *SSR-4 Handle Unknown Filter*

State requirement in the table view above). Other logical subsystems FMEA items are analyzed in the same way as described with the *F-11* FMEA item.

The Filtering System logical subsystems FMEA table contains the following additional column compared with conceptual FMEA tables in the problem domain:

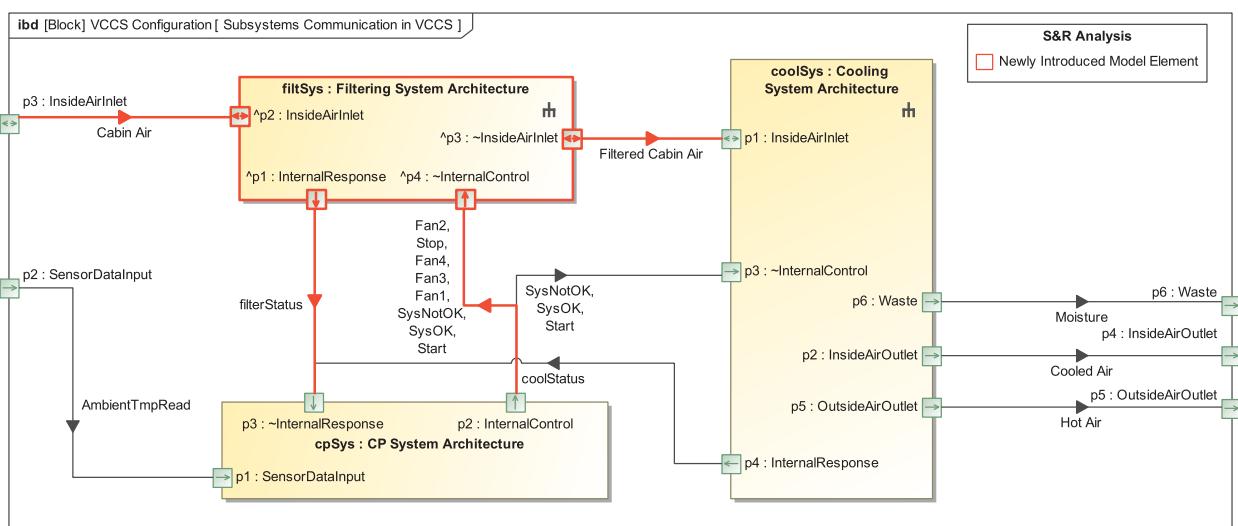
Column name	Purpose
Classification	The classification of failures (FMEA Items) by certain aspects of a system.

The newly identified *SSR-4 Handle Unknown Filter State* safety requirement (depicted in the following figure) can be addressed at the Filtering System.

#	△ Name	Text
1	SSR-4 Handle Unknown Filter State	Provide robust detection for situation where filter state can not be determined due to sensor failure; Safely Stop HVAC function. Provide indicator for the user to visit the service.

Addressing safety and reliability concerns at the configuration level

So, let's jump to the final phase: adapting the configuration model according to the changes introduced in the safety & reliability analysis. Since the filtering subsystem architecture is completed, it should be integrated by ensuring successful interaction with already defined subsystems. For this reason, the interface compatibility verification is performed by specifying the interactions within the particular configuration of the *Sol*.



As you can see, no incompatible interfaces were identified, and we can state that the Filtering System can successfully communicate within the VCCS.

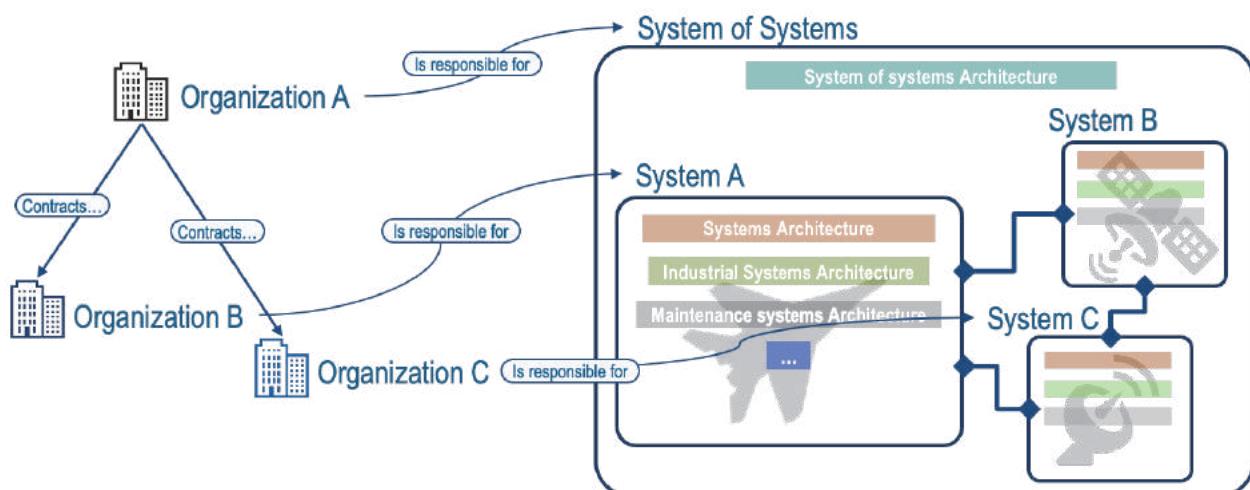
ANNEX B: FROM SYSTEM OF SYSTEMS TO SYSTEM ARCHITECTURE

By Aistė Aleksandravičienė, Gintarė Kriščiūnienė, Aurelijus Morkevičius, PhD

Model-based engineering practices have recently evolved beyond engineering a single system. In this age of innovation, system thinking can be applied in the **system of systems (SoS)** environment to deal with new challenges, such as autonomous traffic control, industry 4.0, the internet of things, etc.

The **SoS** is a **Sol**, whose elements are systems themselves; multiple organizations are involved in design, maintenance, and operation of the **SoS**.

Therefore, when designing a **SoS** it is important to establish a digital environment for multiple organizations to interact, especially when transitioning from **SoS** to **systems architectures (SAs)** (see the following figure). In most cases this is a collaboration between the contracting authority and a contractor, or the Original Equipment Manufacturer (OEM) and a supplier. This annex provides a detailed approach for ensuring the digital continuity from **SoS** to **SAs** in the model-based environment.



System of Systems Engineering

Knowledge of an **SoS** can be captured using the **Unified Architecture Framework® (UAF®) v1.1**, whose layout (also known as a **UAF** grid) is organized into rows and columns, where rows are domains and columns are model kinds. The intersection of a row and column is called a view specification or just a viewpoint.

In this annex, we only focus on the UAF Resources domain, which captures logical **SoS** architectures and how they serve in the achievement of **SoS** capabilities. However, the **UAF** does not specify ways and means to define the architecture for a single system. For this, we utilize MagicGrid with SysML.

	Taxonomy Tx	Structure Sr	Connectivity Cn	Processes Pr	States St	Interaction Scenarios Is	Information If	Parameters Pm	Constraints Ct	Roadmap Rm	Traceability Tr		
Metadata Md	Metadata Taxonomy Md-Tx	Architecture Viewpoints ^a Md-Sr	Metadata Connectivity Md-Cn	Metadata Processes ^a Md-Pr	-	-			Metadata Constraints ^b Md-Ct		Metadata Traceability Md-Tr		
Strategic St	Strategic Taxonomy St-Tx	Strategic Structure St-Sr	Strategic Connectivity St-Cn	-	Strategic States St-St	-			Strategic Constraints St-Ct	Strategic Deployment, St-Rm Strategic Phasing St-Rm	Strategic Traceability St-Tr		
Operational Op	Operational Taxonomy Op-Tx	Operational Structure Op-Sr	Operational Connectivity Op-Cn	Operational Processes Op-Pr	Operational States Op-St	Operational Interaction Scenarios Op-Is			Operational Constraints Op-Ct	-	Operational Traceability Op-Tr		
Services Sv	Service Taxonomy Sv-Tx	Service Structure Sv-Sr	Service Connectivity Sv-Cn	Service Processes Sv-Pr	Service States Sv-St	Service Interaction Scenarios Sv-Is	Conceptual Data Model, Environment Pm-En		Service Constraints Sv-Ct	Service Roadmap Sv-Rm	Service Traceability Sv-Tr		
Personnel Pr	Personnel Taxonomy Pr-Tx	Personnel Structure Pr-Sr	Personnel Connectivity Pr-Cn	Personnel Processes Pr-Pr	Personnel States Pr-St	Personnel Interaction Scenarios Pr-Is	Logical Data Model,		Competence, Drivers, Performance Pr-Ct	Personnel Availability, Personnel Evolution, Personnel Forecast Pr-Rm	Personnel Traceability Pr-Tr		
Resources Rs	Resource Taxonomy Rs-Tx	Resource Structure Rs-Sr	Resource Connectivity Rs-Cn	Resource Processes Rs-Pr	Resource States Rs-St	Resource Interaction Scenarios Rs-Is	Physical Data Model, Measurements Pm-Me		Resource Constraints Rs-Ct	Resource evolution, Resource forecast Rs-Rm	Resource Traceability Rs-Tr		
Security Sc	Security Taxonomy Sc-Tx	Security Structure Sc-Sr	Security Connectivity Sc-Cn	Security Processes Sc-Pr	-	-			Security Constraints Sc-Ct	-	Security Traceability Sc-Tr		
Projects Pj	Project Taxonomy Pj-Tx	Project Structure Pj-Sr	Project Connectivity Pj-Cn	Project Processes Pj-Pr	-	-			-	Project Roadmap Pj-Rm	Project Traceability Pj-Tr		
Standards Sd	Standard Taxonomy Sd-Tx	Standards Structure Sd-Sr	-	-	-	-			-	Standards Roadmap Sd-Rm	Standards Traceability Sd-Tr		
Actuals Resources Ar		Actual Resources Structure, Ar-Sr	Actual Resources Connectivity, Ar-Cn		Simulation ^b				Parametric Execution/ Evaluation ^b	-	-		
Dictionary * Dc													
Summary & Overview Sm-Ov													
Requirements Req													

Transitioning from SoS to SA

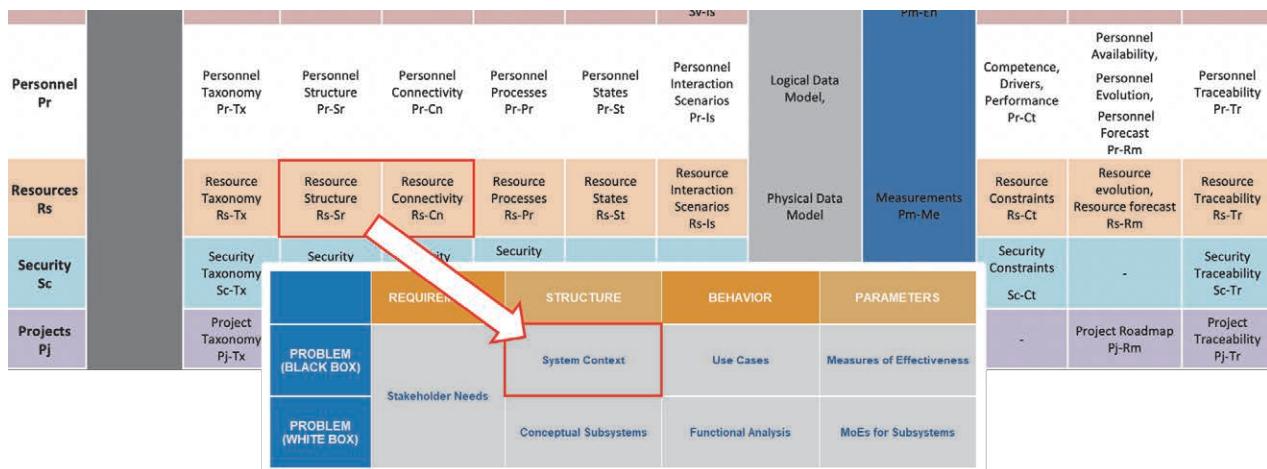
The **UAF** Resources domain captures how different resources, including human resources and systems, interact to implement operational needs and achieve capabilities of the SoS. Resources are grouped into logical containers called capability configurations. Each capability configuration traces to the capability it is meant to achieve at a specific period in time. There might be multiple capability configurations achieving the same capability in different periods of time. At the same time, there might be several alternative capability configurations. It is important to identify which version of the capability configuration to use as an input to analyze the system context of the **Sol**.

MagicGrid: System Context

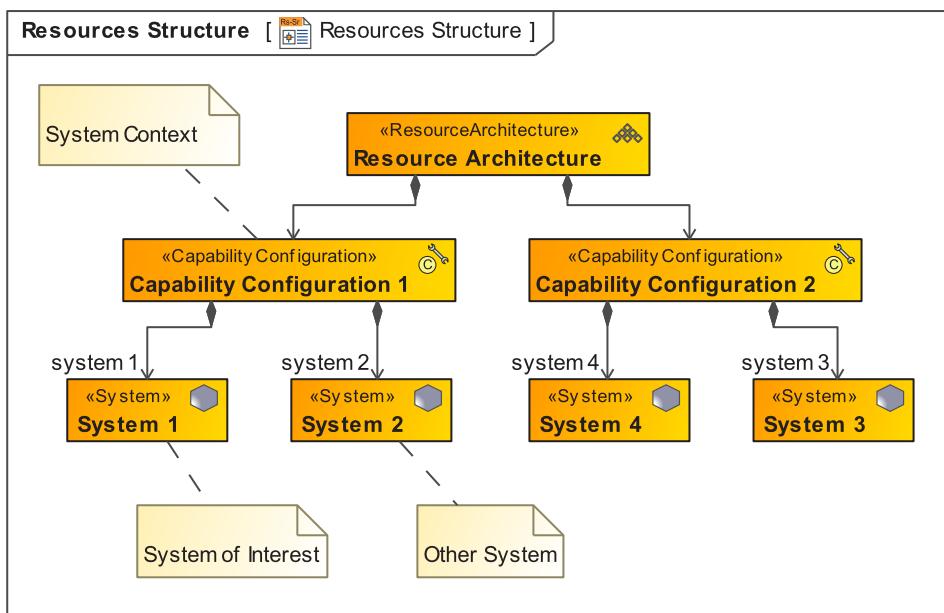
The system context or operating environment determines the external view of the **Sol**. Therefore, the collection of elements in a particular system context includes the **Sol** itself and external systems (natural or artificial), as well as users (humans, organizations) that interact with the **Sol** to exchange data, matter, energy, or human resources.

As defined by MagicGrid, the system context is captured in the model as a SysML block. The **Sol**, external systems, and supposed users are also captured as SysML blocks. While composite association relationships enable modelers to relate them to a particular system context, the SysML internal block diagram provides the infrastructure to display the participants of the system context and their interactions.

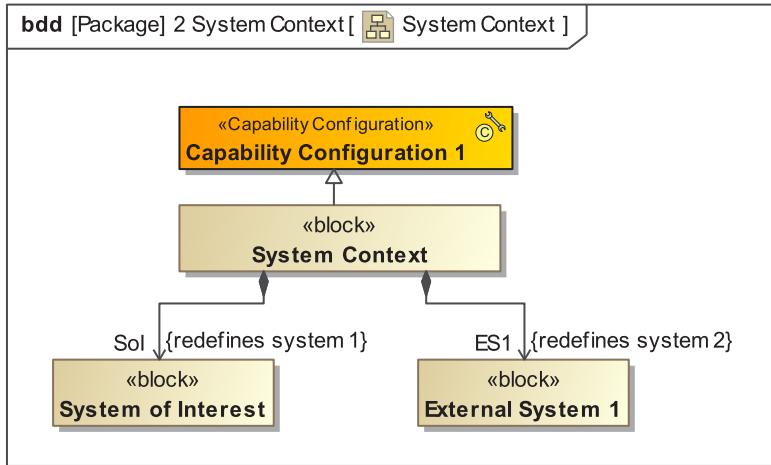
Let's see how to retrieve the system context of a single **Sol** from the **SoS** model. The Resources Structure and Resources Connectivity viewpoints serve as the main source of information in this case (see the following figure).



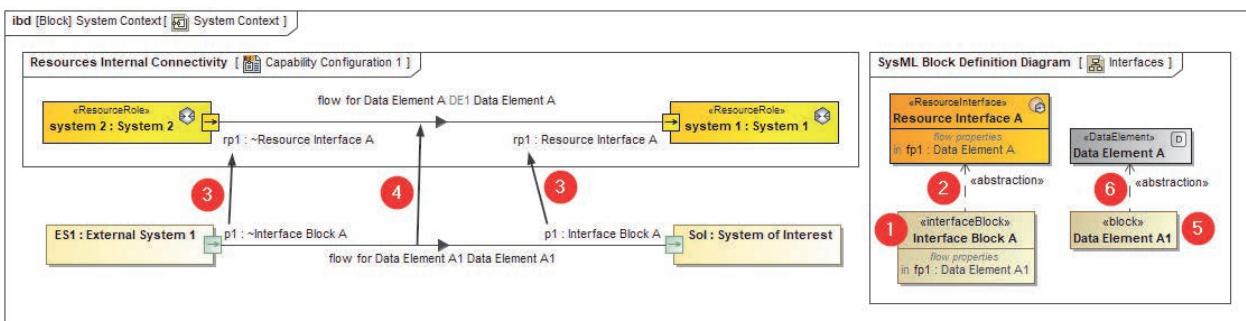
The system context of the **Sol** is initially defined in the **SoS (UAF)** model as the Resources Structure viewpoint, which captures that **Sol** along with other systems within the same **SoS**. The System Context view in the SysML model can be considered a subset of the Resource Structure viewpoint (see the following figure).



Therefore, the block that captures the system context of the particular **Sol** in the **SA (SysML)** model becomes a subtype of the capability configuration in the **SoS (UAF)** model. As a consequence, that block inherits all the parts of the related capability configuration. The parts that participate in the system context of the Sol are redefined with SysML blocks that capture copies of the **Sol**, actors, or external systems in the SysML model (see the following figure). The redefinition of parts is necessary for further modeling: this enables a modeler to update the system context information without modifying the **SoS (UAF)** model. For example, you may need to add new ports or more specific ones than those defined in the **SoS (UAF)** model.



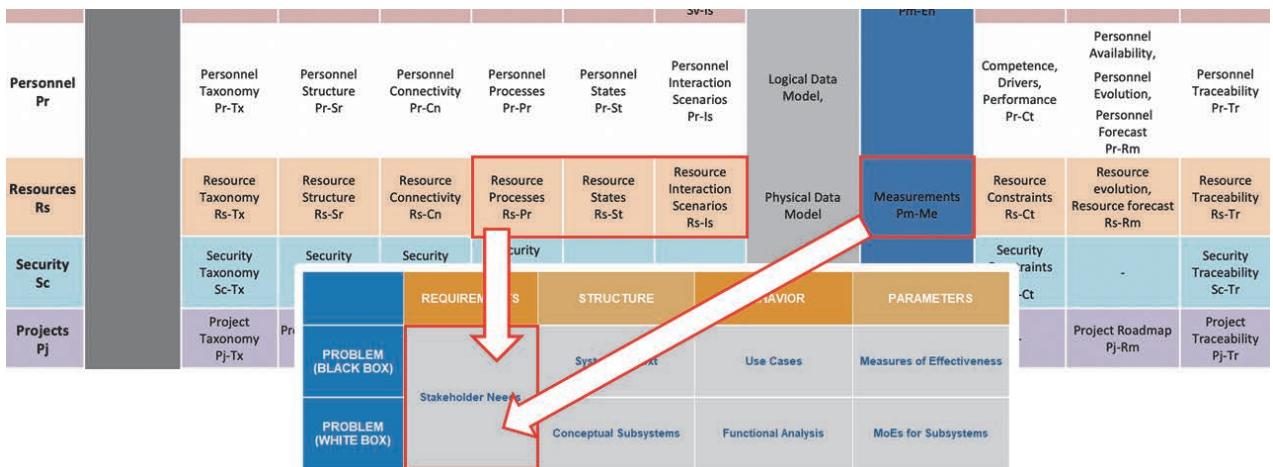
If there are ports/interfaces and interactions captured in the Resources Connectivity viewpoint of the **SoS (UAF)** model, they show up in the **SA (SysML)** model, too. As shown in the following figure, this task includes: (1) creation of relevant SysML interface blocks; (2) creation of the abstraction relationships from the SysML interface blocks to UAF resource interfaces; (3) redefinition of UAF resource ports with SysML proxy ports; and (4) redefinition of UAF resource connectors with SysML connectors. UAF elements (e.g., Data Elements) that capture the items to flow over resource connectors can be reused in the SysML model, unless it requires more specific information. In that case, a new SysML element is created (5), and the abstraction relationship pointing from it to the relevant element in the **UAF** model is specified (6).



MagicGrid: Stakeholder Needs

Stakeholder needs may include primary user needs, system-related government regulations, industry standards, policies, procedures, internal guidelines, etc. They can be captured as SysML requirements, which enable modelers to store the textual information in the **SysML** model. Stakeholder needs can be grouped by the source they came from and categorized into functional and non-functional needs. The infrastructure of the SysML requirements diagram or table can be utilized to display their hierarchy.

Stakeholder needs can be captured by interviewing stakeholders, giving them questionnaires, discussing needs in focus groups, or studying documents written in diverse formats. In addition to these information sources, the **UAF** model of the **SoS** can serve as input to the stakeholder needs as well. To be more precise, the Resource Processes, Resource States, and Resource Interaction Scenarios Viewpoints can be used for identifying functional stakeholder needs, while the Resources Measurements Viewpoint can be used for non-functional (see the following figure).

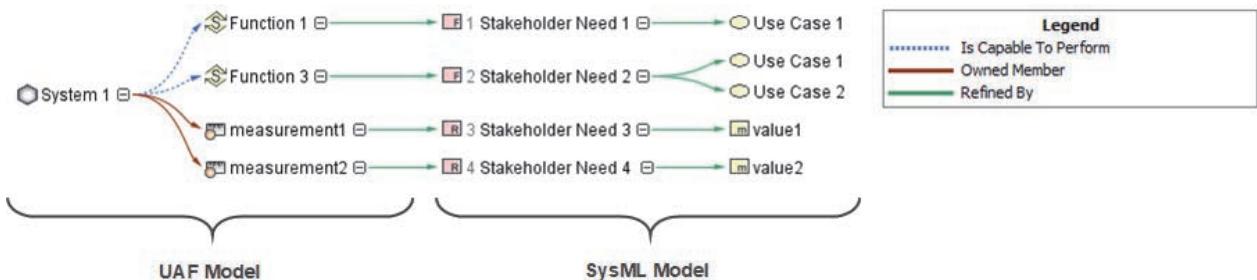


The subset of functions defined in the Resource Processes Viewpoint that are allocated (using the Is Capable To Perform relationship, as shown in the following figure) to the system which becomes the **Sol** in the SysML model motivate the creation of one or more functional stakeholder needs for that **Sol**. This also applies to the case of states and interactions allocated to the relevant system in the **UAF** model.

When functional stakeholder needs are captured, SysML refine relationships can be established from SysML requirements that capture the stakeholder needs to the relevant functions, states, or interactions in the **UAF** model (see the following figure).

Measurements that are owned by the system that becomes the **Sol** in the SysML model motivate the creation of one or more non-functional quantitative stakeholder needs for that **Sol**. When these are captured, SysML refine relationships can be established from them to the relevant measurements in the **UAF** model (see the following figure).

When use cases and **MoEs** of the Sol are captured in the **SysML** model, SysML refine relationships can help to convey which stakeholder needs these use cases and **MoEs** refine.



AFTERWORD

Thank you for taking the time to read the second edition of the MagicGrid BoK. Your feedback on the book is more than welcome. All comments will be considered before releasing the third edition of the book. Additionally, you can sign-up as a reviewer for the next editions of the MagicGrid BoK. Please express your interest by sending an email to **NoMagic.MagicGrid@3ds.com**.

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 white-box 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 an actor to capture these concepts, for the reasons described in [Use Cases](#).

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 black-box 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 white-box perspective, the functional decomposition of the system is performed in order to identify conceptual subsystems (also referred as functional blocks in other methodologies).

Block

A SysML element of definition that 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 a 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 that 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 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 a dashed line with an open arrowhead.

Decision/Merge

The decision node marks the start of an 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 that case, the connector is decorated with a filled-in triangle.

Failure Mode & Effects Analysis (FMEA)

The process of reviewing as many components, assemblies, and subsystems as possible to identify potential failure modes in a system and their causes and effects.

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 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. In MagicGrid, they are called conceptual subsystems.

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 conceptual subsystems (also referred as functional blocks in other methodologies).

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.

Logical System Architecture (LSA) model

The core of a solution domain model. Based on the results of the problem domain analysis, it defines the logical subsystems of the system of interest (as a single-level hierarchy). The purpose of the LSA model is to identify work packages, one for each subsystem, and appoint them to separate engineering teams.

Logical Subsystem Architecture (LSSA) model

A solution domain model that contains the logical architecture of a single logical subsystem and the high-level design of its components, as well as even more precise units.

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.

Measure of Effectiveness (MoE)

A characteristic of the Sol 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 a document-based information exchange.

Modeling tool

CATIA Magic software for MBSE:

- Magic Cyber Systems Engineer / Cameo Systems Modeler along with the Cameo DataHub plugin installed
- Magic System of Systems Architect / Cameo Enterprise Architecture along with the Cameo DataHub plugin installed
- Magic Software Architect / MagicDraw with the SysML and Cameo DataHub plugins

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. Packages can be used to organize the elements into logically cohesive groups.

Parametric diagram

A special type of ibd that displays the internal structure of a block, focusing 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; in other words, a property of a block typed by another block 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 that 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 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.

Reliability

Ability of a functional unit to perform a required function under given conditions for a given time interval (ISO/IEC 2382:2015 Information Technology).

Safety

Freedom from unacceptable risk (IEC 61508:2010 EEPE safety-related systems).

Sequence diagram

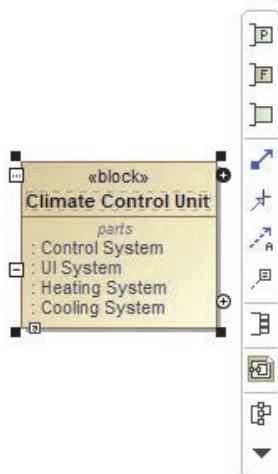
A type of behavioral SysML diagram that can be used to describe how the parts of a block interact with one another via operation calls and asynchronous signals.

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, among others. 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 Sol 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 architecture (SA)

A more abstract, conceptualization-oriented, global architecture, focused to achieve the mission and operational concept of the Sol, in comparison to system design. It focuses on high-level structure in systems and system elements.

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.

System of systems (SoS)

A SoI whose system elements are systems themselves. Typically, these entail large-scale interdisciplinary problems with multiple, heterogeneous, distributed systems.

Systems Engineering (SE)

Interdisciplinary tasks that are required throughout a system's life cycle to transform stakeholder needs, requirements, and constraints into a system solution.

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 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 systems-of-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 an event concurrency.

Unified Architecture Framework (UAF)

Defines ways of representing an enterprise architecture that enables stakeholders to focus on specific areas of interest in the enterprise while retaining sight of the big picture. UAF meets the specific business, operational and systems integration needs of commercial and industrial enterprises as well as the U.S. Department of Defense (DoD), the UK Ministry of Defence (MOD), the North Atlantic Treaty Organization (NATO) and other defense organizations.

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 also be applied).

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. Dandashi, F., Hause, M. C., "UAF for system of systems modeling", 10th System of Systems Engineering Conference (SoSE), pp. 199-204, San Antonio, TX, 2015.
3. 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.omg.sysml.org/View_Paper-IEEE_2013.pdf.
4. Estefan, J. *INCOSE Survey of MBSE Methodologies*, INCOSE TD 2007-003-02, Seattle, WA, USA, 2008.
5. Friedenthal, S., et al. *A Practical Guide to the SysML, 4th Edition: The Systems Modeling Language*. Boston: MK/OMG Press, 2011.
6. Friedland, B., Herrold, J., Ferguson, G., Malone, R. "Conducting a Model Based Systems 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. Hause, M.C., Thom, F., "Bridging the Chasm – Tracing from Architectural Frameworks to SysML", 17th Annual INCOSE International Symposium, pp. 1317-1332, 2007.
9. Hause, M. C., Thom, F., "MoDAF and SysML – A Winning Combination", INCOSE UK Chapter – Spring Symposium 2005, 2005.
10. 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>.
11. ISO/IEC/IEEE 15288:2015 "Systems and Software Engineering–System Life Cycle Processes", Geneva, 2015.
12. 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.
13. Morkevicius, A., Aleksandraviciene, A., Armonas, A., Fanmuy, G. "Towards a Common Systems Engineering Methodology to Cover a Complete System Development Process", 30th Annual INCOSE International Symposium, Cape Town, South Africa, July 2020.
14. 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.
15. 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, pp. 313, Paris, 2015.
16. Morkevicius, A., Jankevicius., N. "An approach: SysML-based automated requirements verification", IEEE International Symposium on Systems Engineering (ISSE), pp. 92-97, Rome, 2015.
17. Object Management Group *OMG Systems Modeling Language (OMG SysML)*, v1.5, May 2017. Available at <https://www.omg.org/spec/SysML/1.5/>.
18. 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/>.
19. Soegaard, S. E. "Adopting MBSE using SysML in System Development–Joint Strike Missile (JSM)", No Magic World Symposium, May 2016. Available at <https://vimeo.com/user28256466/no-magic-world-symposium-2016-presentations/page/3>.
20. Spangelo, S. C. et al. "Applying model based systems engineering (MBSE) to a standard CubeSat", Aerospace Conference IEEE, 2012.
21. 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.
22. Weilkiens, T. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. MK/OMG Press, 2008.
23. Zachman, J. A. "A framework for information systems architecture", IBM Syst. J., vol. 26, no. 3, pp. 276-292, 1987.

ABOUT THE AUTHORS

Aistė Aleksandravičienė

Aistė is an OMG® Certified Systems Modeling Professional (OCSMP). Currently she holds the position of industry business consultant for cyber systems engineering at Dassault Systèmes, CATIA brand.

Aistė takes responsibility for educating Dassault Systèmes clients on the three pillars of MBSE: language (SysML), method (MagicGrid), and tool (CATIA Magic software and integrations). While working with 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. Aistė also works on training material and writes papers (alone and in collaboration with her colleagues) to spread the MBSE culture in the systems engineering community. She is a speaker at multiple MBSE conferences and other public events.

Aistė holds a master's degree in Information Systems Engineering from Kaunas University of Technology (Lithuania).

Aurelijus Morkevičius

Aurelijus is an OMG® Certified UML, Systems Modeling, and BPM Professional. Currently he is a senior manager of industry business consultants for cyber systems engineering at Dassault Systèmes, CATIA brand.

He has expertise with enterprise, mission, model-based systems and software engineering (mostly based on UAF, SysML, and UML accordingly) and defense architectures (DoDAF, MODAF, NAF). Aurelijus is a co-chairman and the leading architect for the current OMG UAF (previously known as UPDM) standard development group. He represents the CATIA brand at INCOSE and NATO.

Aurelijus holds a PhD in Informatics Engineering from Kaunas University and Technology since 2013. He currently teaches an Enterprise Architecture course at the same university. Aurelijus is also an author of multiple articles, and a speaker at multiple conferences.

Aistė Aleksandravičienė
Aurelijus Morkevičius, PH.D.

MagicGrid® BOOK OF KNOWLEDGE
A Practical Guide to Systems Modeling using
MagicGrid from Dassault Systèmes
2nd edition

*Language editor Kim Papke
Cover pictures www.3DS.com
Designed by Arūnas Aleksandravičius*

*Published and printed by
Vitae Litera, UAB
Savanoriu av. 137
LT-44146 Kaunas, Lithuania
www.tuka.lt | info@tuka.lt*



ISBN 978-609-454-554-2

A standard linear barcode is located in the bottom right corner of the page. It is oriented vertically and contains several vertical bars of varying widths.

9 786094 545542