ERJU SYSTEM PILLAR

Systems Engineering Management Plan - Annex M2 Viewpoint Guidelines

Systems Engineering Management Plan - Annex M2 Viewpoint Guidelines

Author(s)	Jorge Block , Dennis Kunz , EDDOUS Sayfeddine (SNCF RESEAU / Directions Techniques Réseau / DGII DTR GE SF Solutions)
Abstract	This document contains the Capella Viewpoints' creation and usage guidelines intended to System Pillar. It also provides the diagram rules, explanations and examples for modeling purposes.
Config Item	System Engineering Management Plan
Document ID	SEMP Annexes/SEMP Annex M2 Viewpoint guidelines#722433 Systems Engineering Management Plan - Annex M2 Viewpoint Guidelines
Classification	Public
Status	Released
Version	1.0
Revision	722433
Last Change Date	02.10.2025
Copyright	Brussels: Europe’s Rail Joint Undertaking, 2025

© Europe’s Rail Joint Undertaking, 2025

This document is drafted by and belongs to EU Rail.

EU Rail encourages the distribution and re-use of this document, the technical specifications and the information it contains. EU Rail holds several intellectual property rights, such as copyright and trade mark rights, which need to be considered when this document is used.

EU Rail authorises you to re-publish, re-use, copy and store this document without changing it, provided that you indicate its source and include the following: EU Rail trade mark, title of the document, year of publication, version of document.

EU Rail makes no representation or warranty as to the accuracy or completeness of the information contained within these documents. EU Rail shall have no liability to any party as a result of the use of the information contained herein. EU Rail will have no liability whatsoever for any indirect or consequential loss or damage, and any such liability is expressly excluded.

You may study, research, implement, adapt, improve and otherwise use the information, the content and the models in the this document for your own purposes. If you decide to publish or disclose any adapted, modified or improved version of this document, any amended implementation or derivative work, then you must indicate that you have modified this document, with a reference to the document name and the terms of use of this document. You may not use EU Rail’s trade marks or name in any way that may state or suggest, directly or indirectly, that EU Rail is the author of your adaptations.

EU Rail cannot be held responsible for your product, even if you have used this document and its content. It is your responsibility to verify the quality, completeness and the accuracy of the information you use, for your own purposes.

Document History

0.1 19.03.2025	Jorge Block	Reviewed version (initial) (initial)
0.2 27.08.2025	EDDOUS Sayfeddine (SNCF RESEAU / Directions Techniques Réseau / DGII DTR GE SF Solutions)	Reviewed version including Findings from Review 0.3
0.3 27.08.2025	EDDOUS Sayfeddine (SNCF RESEAU / Directions Techniques Réseau / DGII DTR GE SF Solutions)	Reviewed version including Findings from Review 0.2
0.2 27.08.2025	EDDOUS Sayfeddine (SNCF RESEAU / Directions Techniques Réseau / DGII DTR GE SF Solutions)	Reviewed version including Findings from Review 0.3

1 19.09.2025	EDDOUS Sayfeddine (SNCF RESEAU / Directions Techniques Réseau / DGII DTR GE SF Solutions)	Approved version based on Review 0.2
1.0 24.09.2025	EDDOUS Sayfeddine (SNCF RESEAU / Directions Techniques Réseau / DGII DTR GE SF Solutions)	Approved version based on Review 1

Approval by reviewers (captured at end of 'In Review by System Pillar')

Comments	**#1** by **EDDOUS Sayfeddine (SNCF RESEAU / Directions Techniques Réseau / DGII DTR GE SF Solutions)** on 2025-09-24 09:51 Due date of approval reached, no answer for remaining waiting approvers.
Approvals	ANTOONS Gilles : Approved , Renard, Marie Pierre (SMO RI MT FR ADC TGMTR3) : Approved , Pérez Gómez-Tostón, Lidia : Waiting , LOEFFLER Christian : Waiting , Schöni Ulrich (I-NAT-GST-CCS) : Waiting , MARTINEZ Laura : Waiting , Morman Bettina (I-NAT-GST-CCS) : Waiting , Marjan Haj-Mohammad-Ali-Hidalgo : Approved , Philipp Wolber : Waiting , Filip Giering : Waiting , MOTTOLA, Giuseppe Diodato : Waiting , michael.wild : Waiting , Simon Lambert : Waiting , Thomas Laguerie : Waiting , Ibtihel Cherif : Waiting , GAUTRON Mickael (SNCF VOYAGEURS / DIRECTION DE L'INGENIERIE DU MATERIEL / CIM - SBF 1) : Waiting , Steffens, Sonja (SMO RI R&D F IL) : Waiting
Type of Approval	Document Review
Attachments	Annex_M2_comments_export.xls

Approval by approvers (captured at end of 'In Approval by System Pillar')

Approvals	Schmidt Steffen (I-NAT-GST-ERTM) : Approved , SCHWAN Nico : Approved , SANGO Marc (SNCF / DIR TECHNOLOGIES INNOVATION ET PROJETS GROUPE / IR DIR RECHERCHE - PSF) : Approved , KEFALAS, Georgios : Approved
Type of Approval	Document Approval

Table of contents

1 Preamble

1.1 Purpose

This guideline focuses on how to create and use viewpoints in System Pillar. Viewpoints help in organising and understanding the different aspects of a complex system, making it easier to design, analyse, and communicate about the system.
In this document, you will find:

- A clear explanation of what viewpoints are and why they are important for the System Pillar.
- Definitions of the different viewpoints and their role in system development.
- Simple guidance on how to apply these viewpoints in order to improve understanding and decision-making.

The purpose of this document is to ensure that every involved engineer, system architect, and stakeholders can look at the system from the right perspectives and work together effectively.
By following this guide, teams can ensure all important aspects of the system are considered and aligned with the goals of the System Pillar.

In addition this guideline provides viewpoints with diagram rules, diagram explanations and examples as well as view-specific rules for each used ARCADIA layer inside System Pillar Reference Architecture model.

For rules and descriptions regarding single elements in the model, please consider *SPPROCESS/10 SEMP V 01_01/SEMP Annex M1 Capella element rules : 722433*.

For analysis methods and quality check principles of these viewpoints, refer to Requirements Management Plan.

1.2 Intended Audience

The content of this document is valid for all the System Pillar tasks and domains. It is intended to Capella modelers and other model users in System Pillar project or outside (refer to SPPR-10950 - Stakeholders adressed).

1.3 Document Context

See purpose.

1.4 Glossary

The definitions are included as descriptions for viewpoint descriptions.

Acronym (abbreviation)	Full text (title)
ARCADIA	(Arc)hitecture (A)nalysis and (D)esign (I)ntegrated (A)pproach
C2P	Capella2Polarion

2 Stakeholders

Stakeholders adressed
This list of stakeholders is generic and applicable to all viewpoints. If specific stakeholders should be considered for a particular viewpoint, they will be mentioned directly in the corresponding section of the viewpoint.
The role/need of each stakeholder regarding a specific viewpoint is detailed in the corresponding section of the viewpoint.

Creator of a viewpoint:

- SPPR-10696 - Systems Engineer as Modeler

Users of a viewpoint:

- SPORG-87 - Environment Engineering Team : Ensure consistency with the modelling methods defined within the SP SEMP
- SPORG-85 - PRAMSS Team : Ensure consistency with the domain policies, methods and principles for PRAMSS aspects
- SPORG-93 - T2 ARC : Ensure functional and technical consistency of the viewpoint with the whole System Pillar Reference Architecture, and validate the viewpoint
- SPPR-2653 - Mirror Group : Analyze the functional and technical feasibility of the viewpoint, the coverage of needs and the impacts on current systems. See Systems Engineering Management Plan - Annex MG Mirror Group Guideline.
- SPPR-11183 - Test Engineer : Use the viewpoint for test purposes to verify the correctness and completeness of the system behaviour with its environment.

3 Viewpoints

3.1 System context description viewpoint

3.1.1 Description, stakeholders and concerns

Description
The list of definitions related to the concept of System Context is listed below:

System Context
Describes the system relationships and environment that exist around a system.

System
Arrangement of system elements, that together exhibit a stated behaviour or meaning that the individual constituents do not.

According to ISO 15288 a system is “a combination of interacting elements organised to achieve one or more stated purposes. “. In terms of this document, a system in black box view is furthermore defined by:

1. interfaces to actors outside the system, defining the system boundary
2. at least one function allocated to it

A system in white box view can be further refined into (exclusive or):

1. into a more granular systems, hence making itself to a system of systems
2. into subsystems on the lowest level of system of systems refinement

In both cases, a system is a conceptual entity that aggregates the properties of its constituents but is not the element that defines the properties itself. A system is hence subject to the emerging properties of its constituents.
Usage context definitions of term „system“:

- Constituent system: according to ISO 21839, a system that forms part of a system of systems
- System of interest: according to ISO 21839, a system whose life cycle or properties are under consideration in a given context

System Of Systems
According to ISO 21839, a system of systems is set of systems that interact to provide a unique capability that none of the constituent systems can accomplish on its own.
A system of systems in our understanding at least comprises two constituent systems.
A system of systems itself can be nested as a constituent system in a larger system of systems, i.e. system of systems can span multiple levels recursively.

Actor
An actor is an entity external to the system under consideration, which can be either a human or a technical system. Actors interact with the system through interfaces.

This definition is applicable to system actor, logical actor, and physical actor.

Viewpoint stakeholders
The list of stakeholders mentioned in SPPR-10950 - Stakeholders adressed is applicable to this viewpoint.

The specific stakeholders/roles addressed by this viewpoint are:

Creator of this viewpoint:

- SPPR-10696 - Systems Engineer as Modeler : Define the system boundaries, align the viewpoint with the system concept for <structural element name> and its related upper level requirements

Users of this viewpoint:

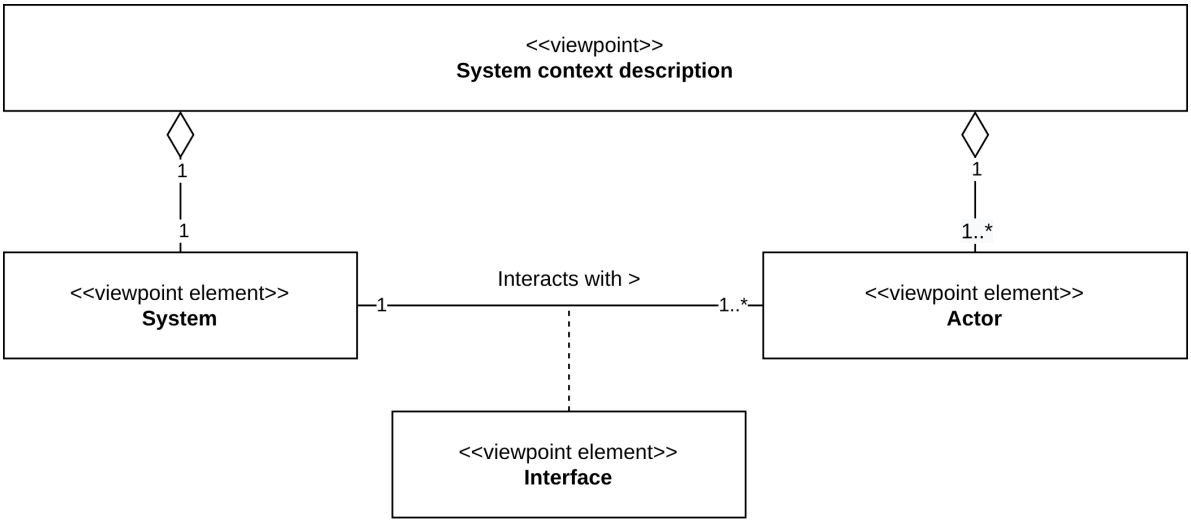
- SPPR-11183 - Test Engineer : Use the system context for integration and test purposes and to verify the correctness and completeness of the system behaviour with its environment.

Concerns
This list of concerns is not exhaustive and can be completed.

- Which external systems or actors are interfaced with the system?
- Are all interfaces and types of exchanges (data, control, material) with external actors identified?
- Is the boundary of the system correctly defined (related capabilities, functions, constraints imposed)?
- Is the level of abstraction appropriate for the analysis level of the system?
- Is the system context aligned with the operational requirements?

3.1.2 Visualisation kind

Meta model of the visualisation kind for the system context description



3.1.3 View method

Construction method system context description
Use Capella Architecture Diagrams to model the system context. The Capella elements to be used in an architecture context viewpoint are:

System analysis

- SPPR-10060 - Actor (System Actor)
- SPPR-10062 - System (System Component)
- SPPR-10071 - Interface (System Component Exchange)

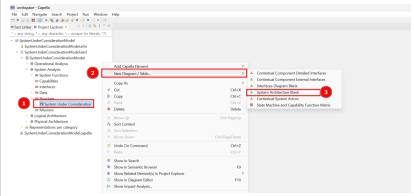
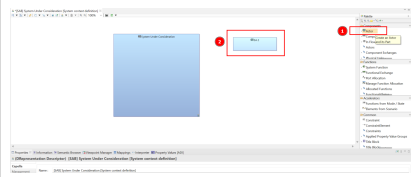
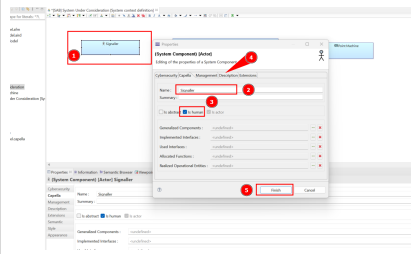
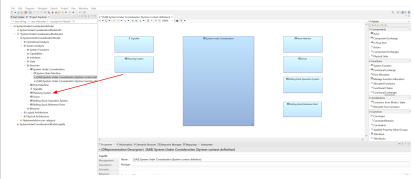
Logical Architecture

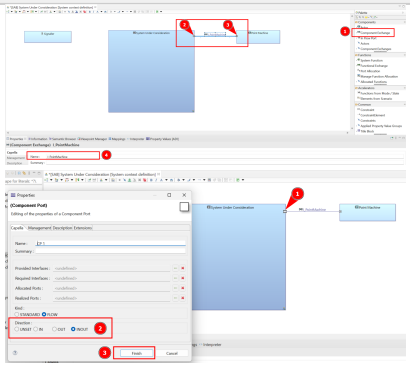
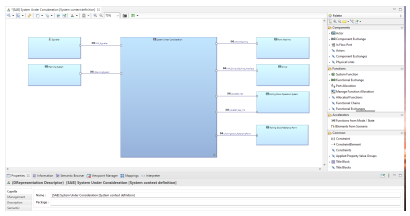
- SPPR-10060 - Actor (Logical Actor)
- SPPR-10062 - System (Logical Component)
- SPPR-10071 - Interface (Logical Component Exchange)

Physical Architecture

- SPPR-10060 - Actor (Physical Actor)
- SPPR-10062 - System (Physical Component (Node or Behaviour)
- SPPR-10071 - Interface (Physical Component Exchange)

Here is a step by step tutorial to create a system context description viewpoint in Capella:

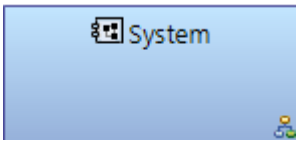
Step description	Illustration
Create in Capella one Architecture Blank diagram e.g. [SAB], to specify participants for the **system context** diagram. To create the diagram: 1. Right-click the **System Under Consideration** 2. Select New Diagram 3. Select Architecture Blank 4. Define the name according to diagram naming convention	
Create system actors in the model	
Capture the first defined system **actor** in the model on the System context definition diagram. To capture participants of the System Under Consideration context: **Adding new actors** 1. Click-left on "Actor" in the Palette 2. Click-left on the white space next to the System Under Consideration. **Adding existing actors** 1. Click-left on "Actors" in the Palette 2. Click-left on the white space next to the System Under Consideration.	
Setting properties of the created actor: 1. Double click-left on the actor on the diagram 2. Include the name of the actor 3. Set is human in case of an human actor or in case of a external system do not select is human. 4. Include the description for the actor according to modelling rules. 5. Click finish and repeat the steps for each actor that needs to be created and defined	
Complete example: Stakeholder needs to indicate that the System Under Consideration system context includes in addition to the Signaller the following actors: • Planning System, Point Machine, Driver, Rolling Stock Operation System, Rolling Stock Reference Point	
Define interfaces in the model	
Capture and define an interface between the system of interest and system actors. Some interfaces could also be defined by constraints found in the Constraint Record. To create component exchange representing the interface: 1. Click-left on component exchange in the palette 2. Click-left on the System Under Consideration 3. Click-left on the System Actor 4. Include name of the component exchange according to Modelling Rules	

Step description	Illustration
Define the direction of the ports: 1. Click-left on one port 2. Click-left UNSET, IN, OUT, INOUT 3. Finish	
Complete the example: Create the other component exchanges represented in example.	

Interpretation method context diagram
The main model elements of an architecture diagram are described below.

Actor

System



Interface

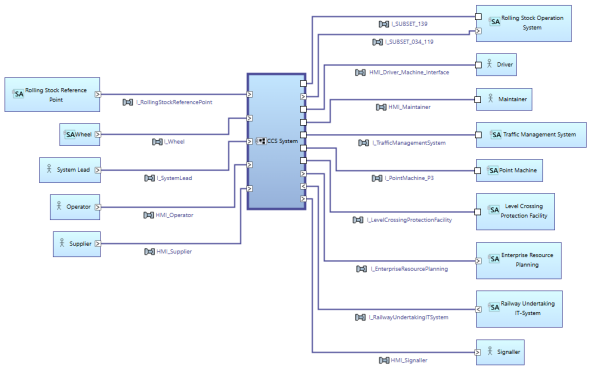
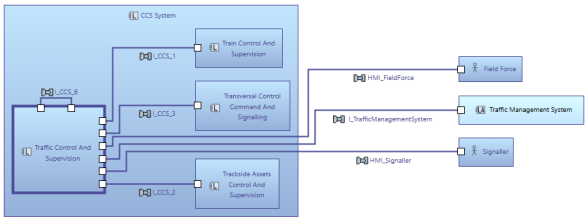
Interaction between system and actor through ports. Port can be input (), output (), or input and output ().

3.1.4 View modelling rules

System context description

1. one diagram exists for the system of interest
2. the system of interest shall be shown
3. all actors shall be shown
4. all component exchanges between the system of interest and the actors shall be shown

Notes	Diagram type abbreviation: [SAB] [LAB [PAB]
ID	SPPR-6427

Example	CCS System  Example Traffic CS System 

3.2 Capability viewpoint

3.2.1 Description, stakeholders and concerns

Description
The list of definitions related to the concept of Capability is listed below:

System Capability
System capabilities define the need to the system of interest to satisfy its stakeholders including how the system capabilities relate to each other and how they relate to actors.

It describes the high-level behaviour of a system and its interaction with other involved entities, resulting in a specific, observable outcome. This capability signifies the system's expected ability in delivering services that represents an operational objective. Essentially, a system capability defines a context for system usage, characterised by a set of exchanges scenarios that outline the conditions under which system functions contribute to achieving the objective.

Traditional approaches like the ARCADIA method describe system capabilities from the system's perspective. However, we've opted to deviate from this method, focusing on an actor's perspective similar to use cases to suit our needs better. This deviation aims to create a stronger distinction between capability and function and ensure compatibility with SysML.

The actor-oriented approach allows the capability to be documented from the perspective of those interacting with the system, enhancing clarity in defining system objectives and functionality. For example, rather than solely focusing on the system's technical aspects, we assess how the system's capabilities meet the needs and objectives of the users or external entities, ensuring comprehensive alignment with stakeholder goals.

Basic elements of a capability

- A capability is a service or a behaviour that your system will perform. The capability name, therefore, is always a verb phrase.
- Capabilities are the subset of system behaviours that external actors can directly invoke or are being involved in.
- An actor can be a person or an external system that interfaces with the system under consideration.
- The capability name does not convey a lot of information. Addition information needs to be defined and modelled for each capability to narrate how your system and its actors collaborate to achieve the capability objective:
    - Description includes:
        - Stakeholder needs/Primary actor needs: someone or something with a vested interest in the behaviour of the system
        - Scope: the entity that owns (provides) the capability (for example, the name of an organisation or system)
        - related information: whatever your project needs for additional information
    - Precondition:
      The state of the system or actors **BEFORE** the capability starts.
      Example: System is in Operation. Train is in initial position.
    - Trigger:
      Event that starts the capability.
      Example: System receives an operational plan from Traffic Management System for moving train from A to B.
      Note: Not used in ARCADIA but helpful for the design.
    - Behavior description:
      What the systems does. Described by exchange scenarios or functional chain involving defined functions and functional exchanges.
    - Postcondition:
      The state of the system or their actors **AFTER** the capability is performed.
      Example: Train is in position B, Position is reported to Traffic Management System.

Note:

- Relationship between the pre- and postcondition of the capability and the sencario that descripes the capability:
    - If the conditions of the owning capability fully applies to the considered scenario, then the conditions are taken from the owning capability.
    - If the conditions of the owning capability do not (fully) apply to the considered scenario, then the precondition is defined suitable for the considered scope of the scenario.

Viewpoint stakeholders
The list of stakeholders mentioned in SPPR-10950 - Stakeholders adressed is applicable to this viewpoint.

The specific stakeholders/roles addressed by this viewpoint are:

Creator of this viewpoint :

- SPPR-10696 - Systems Engineer as Modeler : Define the capability of the <structural element name>, align the viewpoint with the upper system level capabilities

Users of this viewpoint :

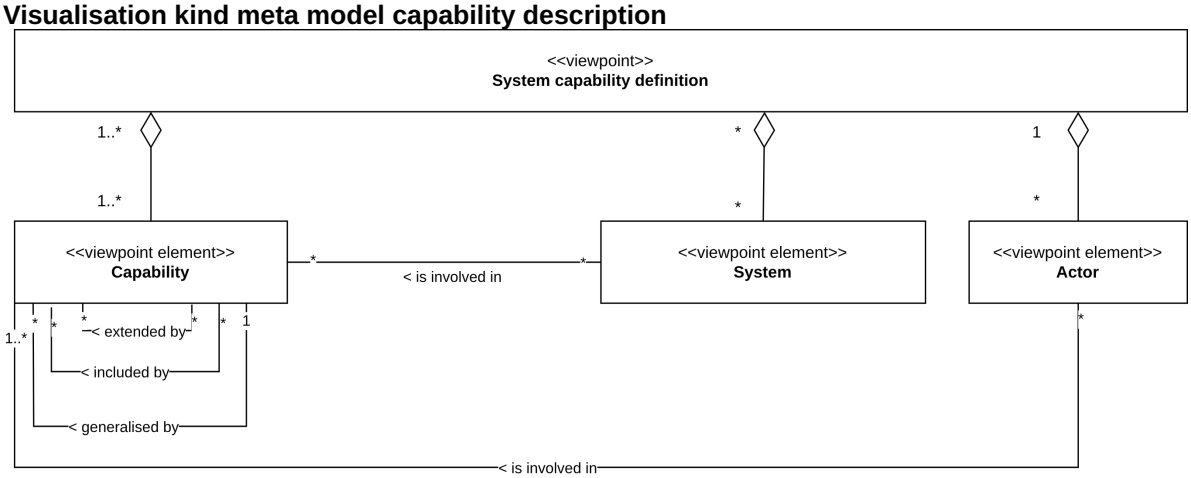
- SPPR-11183 - Test Engineer : Use the capability viewpoint to verify the correctness and completeness of the system behaviour with regards to its requirements

Concerns
This list of concerns is not exhaustive and can be completed.

- Does the capabilities meet the operational and business needs?
- Are the capabilities well structured?
- Are the dependencies between capabilities identified?
- Are constraints (e.g. hardware limitations, SW, etc) and non-functional requirements (e.g. performance, reliability, maintainability, security, etc) considered for the capability overview definition?
- Do the capabilities relate to and reflect the upper level capabilities and the CBO?
- Is the level of detail of the capability appropriate for the analysis level of the system?

3.2.2 Visualisation kind

Visualisation kind meta model capability description



3.2.3 View method

Construction method capability definition
Use Capella Capability Context or Definition diagrams to model the capability description. See SPPR-10822 - View modelling rules.

Capella elements to be used in a capability viewpoint are :

Operational Analysis

- SPPR-10221 - Operational Entity
- SPPR-10086 - Capability (Operational Capability)

System Need Analysis

- SPPR-10060 - Actor (System Actor)
- SPPR-10086 - Capability (System Capability)

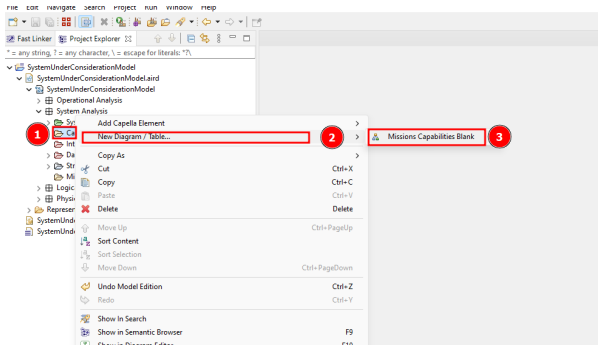
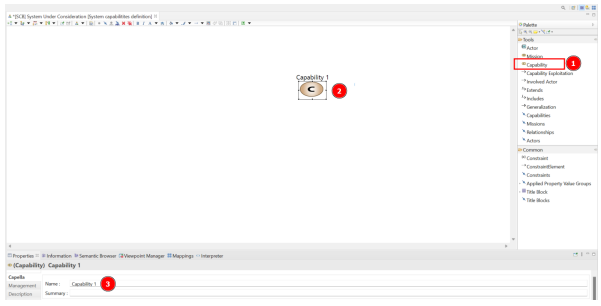
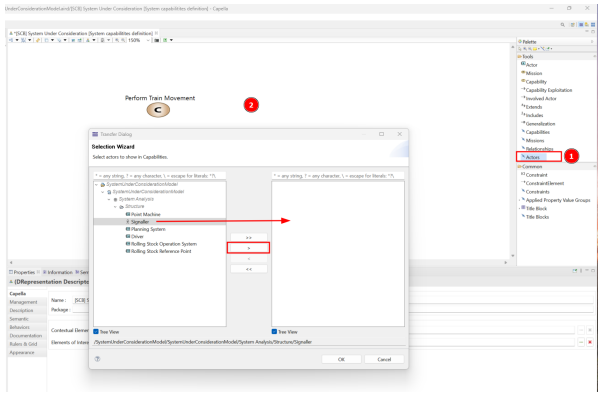
Logical Architecture

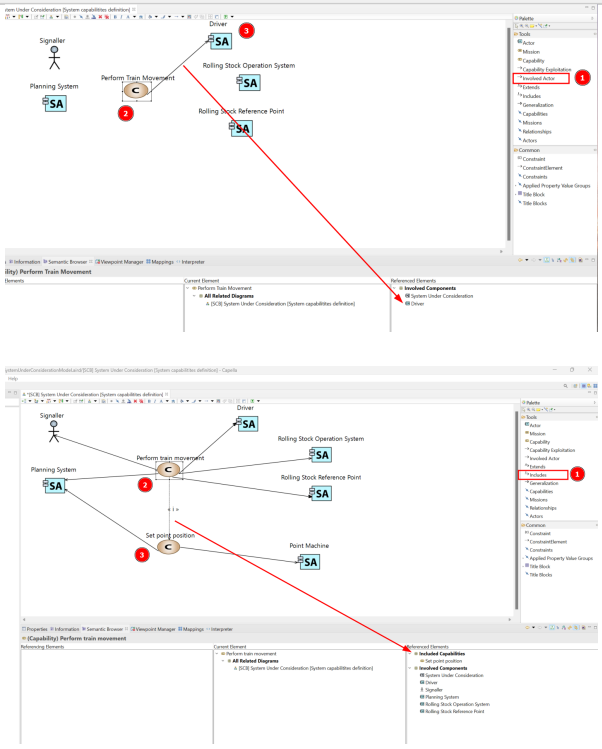
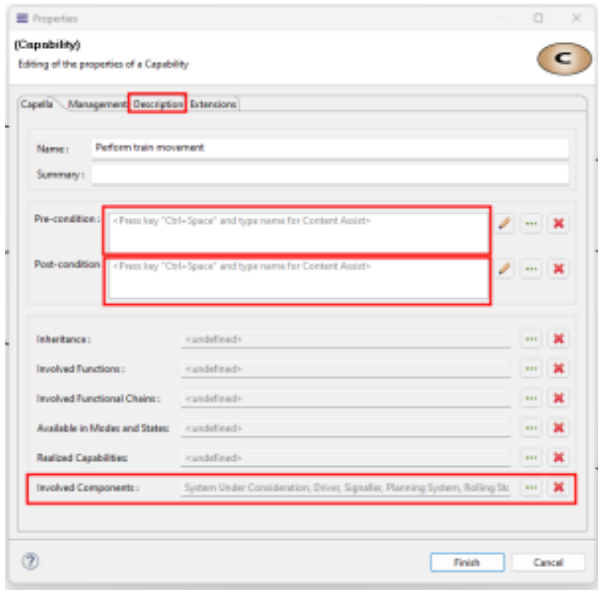
- SPPR-10060 - Actor (Logical Actor)
- SPPR-10062 - System (Logical Component)
- SPPR-10086 - Capability (Logical Capability Realisation)

Physical Architecture

- SPPR-10060 - Actor (Physical Actor)
- SPPR-10062 - System (Physical Component (Node or Behaviour))
- SPPR-10086 - Capability (Physical Capability Realisation)

Here is a step by step tutorial to create a Capability viewpoint in Capella:

Step description	Screenshot
Create in Capella one Mission Capability Blank [MCB] diagram, to represent stakeholder needs from a capability context. To create the diagram 1. Right-click the package capability 2. Select New Diagram 3. Select **Mission Capability Blank** 4. Define the name according to diagram naming convention Alternative: Create capability context diagram.	
Capture the first defined system capability in the model on the System capabilities definition diagram. To capture the capabilities: 1. Click-left on capabilities in the palette 2. Click-left on the white space. 3. Include a proper name. The capability name, is always a verb phrase.	
Include already defined actors in the diagram (If needed new actors can be also created in the capability diagram) 1. Click-left on the blue \ to include actors, that already exists in the model, on the diagram 2. Select the actors for the diagram and move them to the right 3. Ok	
Define capability relationships (SPPR-7868 - Interpretation method capability relationships)	

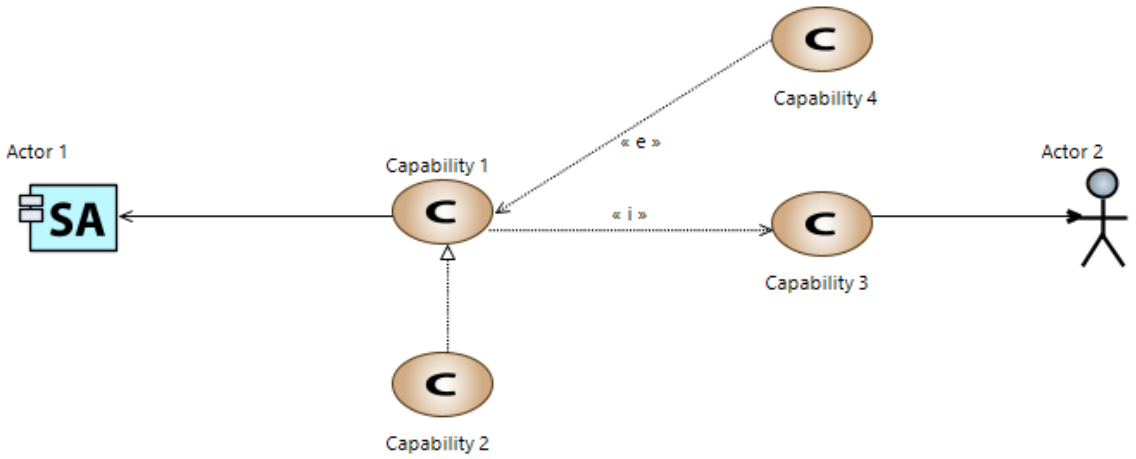
Step description	Screenshot
	
Define capability attributes Description • Stakeholder needs/Primary actor needs: someone or something with a vested interest in the behaviour of the system • Scope: the entity that owns (provides) the capability (for example, the name of an organisation or system) • related information: whatever your project needs for additional information Pre-condition The state of the system or actors BEFORE the capability starts. Example: System is in Operation, Train is in initial position. Post-condition The state of the system or their actors AFTER the capability is performed.	

How to consolidate a set of capabilities
The following points needs to be considered to comprise the analysis of a set of capabilities.

- **Grouping:** Organising the capabilities into groups will bring structure to the views and facilitate understanding. The groups are functional nature and/or depend on stakeholder allocation. Set relations (include, extend, generalise) between the capabilities if required.
- **Duplicates:** Where duplicate capabilities exist, they should be eliminated. Ensure that a duplicate is replaced by the master item on all views where it appears, BEFORE deleting the duplicate. On deletion, the duplicate should no longer be related to any other model element.
- If a deletion is not reasonable an inclusion, extension or generalisation relation between the capabilities should be considered. If the capabilities just "sound" similar, but have different meanings, their names should be changed.

- **Gaps:** Gaps between the capabilities should be checked to ensure that nothing important has been missed. When there are obvious gaps identified the missing capabilities should be added. Nevertheless, the root cause of the lack should be identified to make sure this capability is required and was not accidentally omitted.
- **Naming:** The names of the capabilities must be consistent and according to the modelling rules.
    - SPPR-3986 - Capability has an unique name
- **Overlaps:** The capabilities start conditions corresponding to the end conditions of other capabilities (and vice versa). Overlaps should be avoided. This may results in updating the pre- and postcondition

Interpretation method capability relationships
Capability diagrams are available at every engineering phase in ARCADIA, but for example particularly useful for system analysis. Two kinds of diagrams can be created for capabilities either the context of one capability of interest (COC, CC, CCRI) or the consolidated capability diagram (MCB, CRB).
The context of **one capability diagram** shows only one capability of interest with its relationships to other capabilities and its involved actors. The **consolidated capability diagram** shows multiple capabilities and all their relationships. The consolidated capability diagrams should group capabilities on a similar theme.



Involvement relationships
This is used to relate capabilities to actors or components, "Capability 1" involves "Actor 2"

Include relationship
Is used when a piece of functionality may be split from the main capability, for example to be used by another capability. A simple way to think about this is to consider the included capability as always being part of the parent capability. This is used to try to spot common functionality among capabilities. It is highly possible that one or more of the decomposed capabilities may be used by another part of the system.
It is shown using a dashed line with an open arrow head, the line bearing the «i». The direction of the arrow should make sense when the model is read aloud. As in the diagram the base "Capability 1" includes "Capability 3".

When one capability includes another capability with the "include"-relationship then the scenarios of the included one are referenced in the other capability, the functional chains of the included one are referenced in the other capability

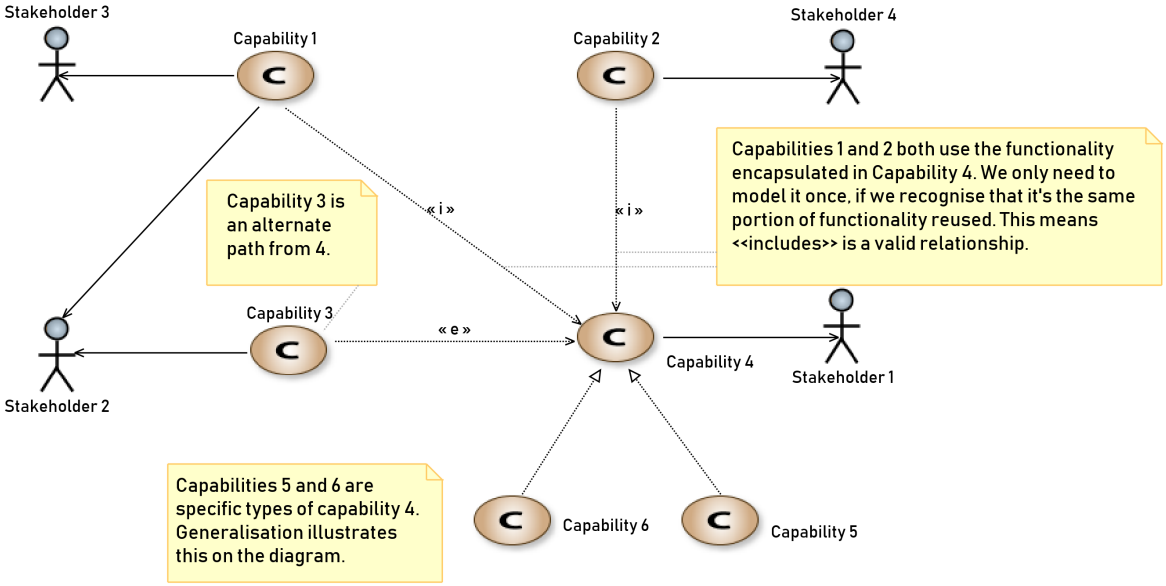
Extend relationship
Is used when the functionality of the base capability is being extended in some way. A simple way to think about this is to consider the extending capability as sometimes being part of the parent capability, which its functionality may change, depending on what happens when the System is running. Extending capabilities are often used to capture special, usually error-handling, behaviour.

The extend relationship is also shown using a dashed line with an open arrow head, the line bearing the stereotype «e». It is important to get the direction of the relationship correct, as it is different from the «i» direction. As in the diagram the "Capability 4" extends the base "Capability 1".Technically, in a base capability, if there are alternative scenarios that have different post-conditions, different actors involved, triggered by something and could be considered as special behaviour, then may be an extending capability of the base capability should be created. Those extending capabilities have different postconditions, have an observable effect to new actors, considered as error handling behaviour and also share common behaviour. The concrete separation between base capability and extending capability should be done through extension point. Those extension points could be represented as a condition of a decision node of the base capability. If the condition is true, then the Extend action is performed.

When one capability extends another capability with the "extent"-relationship then the scenarios of the extended one are referenced optionally in the other capability (OPT fragment,) the functional chains of the extended one are referenced optionally in the other capability (OR fragment)

Generalisation relationship
A generalisation relationship shall be used to link the former to abstract capability with the refined capability. Generalisation between capabilities allows for inheritance of behaviour and relationships. Specialisation/generalisation is shown by a line with an unfilled triangular arrowhead at one end. Specialisation refers to the case when a capability is being made more special or is being refined in some way. As in the diagram "Capability 2" is a special kind of "Capability 1".

Complex example:
The following example shows a subset of the capabilities that focus on a given aspect and the relationships between them.



The example is written from the perspective of stakeholder 1.

- "Capability 5" and "Capability 6" involve "Stakeholder 1", because they are types of "Capability 4", which has a direct involvement relationship with "Stakeholder 1".
- "Capability 1" and "Capability 2" both involve "Stakeholder 1", because "Capability 4" is included within "Capability 1" and "Capability 2".
- "Capability 3" involves "Stakeholder 1", because "Capability 3" extends "Capability 4", which has a direct involvement relationship with "Stakeholder 1".

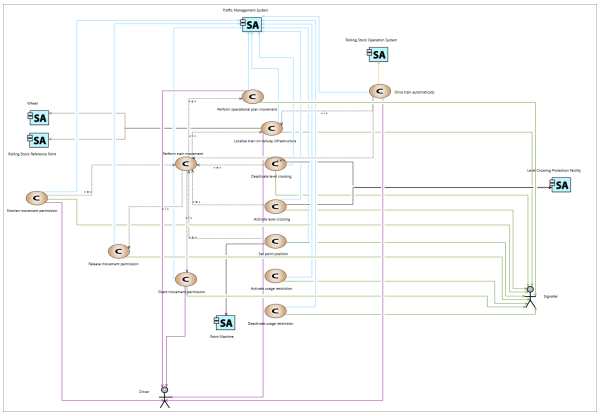
3.2.4 View modelling rules

Capabilities definition

1. at least diagram exists for the system of interest

2. all defined Actors shall be shown
3. all defined capabilities shall be shown
4. all defined relationships between capabilities and actors shall be shown

Actor involvements may be shown to avoid clutter in case of large numbers of actors or capabilities. In case of many capabilities it is a possible option to create multiple diagrams.

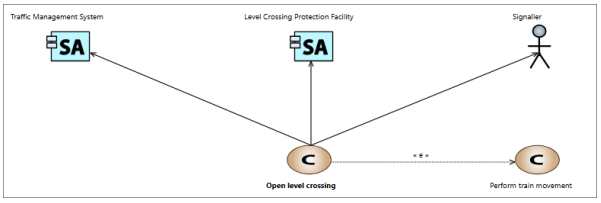
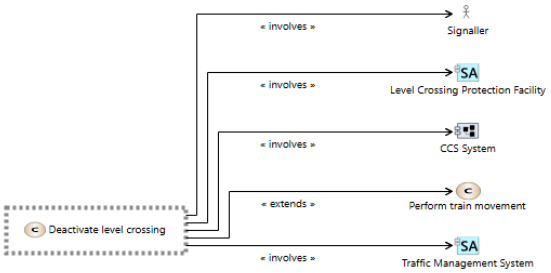
Notes	[SCB] [LCRB]
ID	SPPR-6629
Example	

System capability context
Describes one system capability and how it relates to system actors and other system capabilities.

1. one diagram exists for each capability
2. the considered capability shall be shown in the middle of the diagram
3. all system actors involved in the considered capability are shown
4. all actor involvement relationships shall be shown for the considered capability
5. all relationships to other capabilities shall be shown for the considered capability
6. the name of the considered capability shall be shown in bold

Note: capabilities other than the considered capability may be shown optionally including the relationships and other system actors, if it is useful in a specific case.

Notes	[SCC] [LCR]
ID	SPPR-6630

Example	Manually created in capella:  Auto generated by C2P: 

3.3 Functional flow viewpoint

3.3.1 Description, stakeholders and concerns

Description
The list of definitions related to the concept of function is listed below:

Function
A function is a transformation of inputs into outputs which is purpose-oriented and pays towards a higher goal of a system. All possible input and output values are defined by the exchange items allocated to the functional exchanges of the function. Consequently, when the inputs to the system change, the transformations within the system modify the outputs accordingly. They can be available in a range of system states and therefore change its behaviour based on the current state of inputs.

The structure of functions does not have to reflect any possible implementation and does not have to follow an object decomposition paradigm, as would be followed normally by software engineers implementing one or more systems. In addition, each function is continuously performed by the system or system actors. They are not created, called and terminated.

Functions are allocated to the system or to the system actors. Each function is allocated to one or multiple functional requirements, defining “what” the function is doing. The expected characteristics of functions are then specified via non-functional requirements, which define the “how” (how safe, how accurate, how fast, how reliable, etc.) the function is performing the transformation.

Functional Exchange
A connection between two functions or two components that allows them to convey exchange items.

Viewpoint stakeholders
The list of stakeholders mentioned in SPPR-10950 - Stakeholders adressed is applicable to this viewpoint.

The specific stakeholders/roles addressed by this viewpoint are:

Creator of this viewpoint :

- SPPR-10696 - Systems Engineer as Modeler : Define and refine functional exchanges and dependencies between functions for the corresponding capabilities, and allocate them to interfaces

Users of this viewpoint :

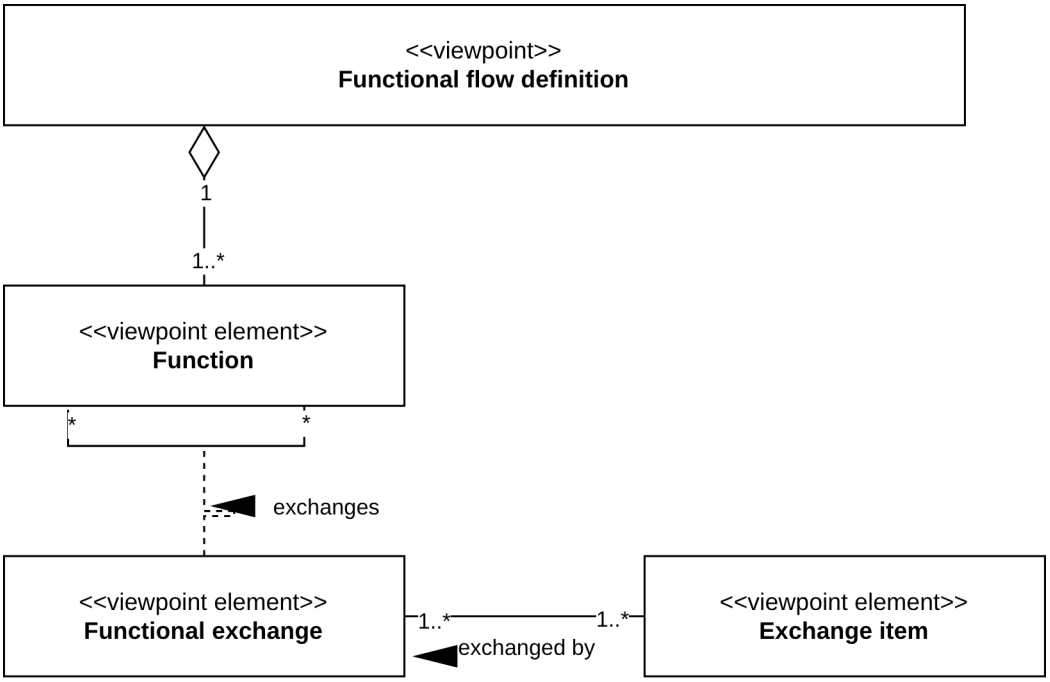
- SPPR-11183 - Test Engineer : Align functional flows with system architecture and validate the system behavior logic, ensure the correct implementation of the functional flows in hardware and software.

Concerns
This list of concerns is not exhaustive and can be completed.

- Are functional flows correctly distributed accross the system, system components or actors' allocated functions?
- Are all functional exchanges crossing components boundaries well allocated in the component exchanges (interfaces) ?
- Are constraints (e.g. hardware limitations, redundancy, etc) and non-functional requirements (e.g. performance, reliability, maintainability, etc) considered for the functional flows definition?
- Is the level of detail of functional flows appropriate for the analysis level of the system?
- Are the control and data flows correctly defined between system components (and actors)?
- How do the functional flows describe the capabilities and interfaces ?

3.3.2 Visualisation kind

Visualisation kind meta model Functional flow definition



3.3.3 View method

Construction method functional flow definition
Use Capella Functional flow diagrams to model the functional flow viewpoints. See SPPR-10850 - View modelling rules. Capella elements to be used in a functional flow viewpoint are:

All Layers

- SPPR-10105 - Exchange Item

Operational Analysis

- SPPR-10220 - Operational Activity
- SPPR-10222 - Operational Interaction

System Need Analysis

- SPPR-10082 - Function (System Function)
- SPPR-10085 - Functional Exchange (System Functional Exchange)

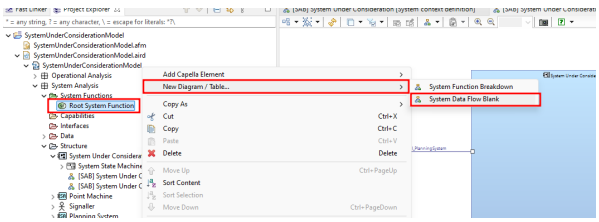
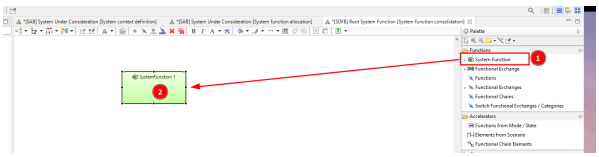
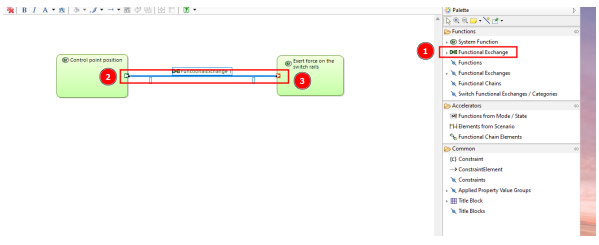
Logical Architecture

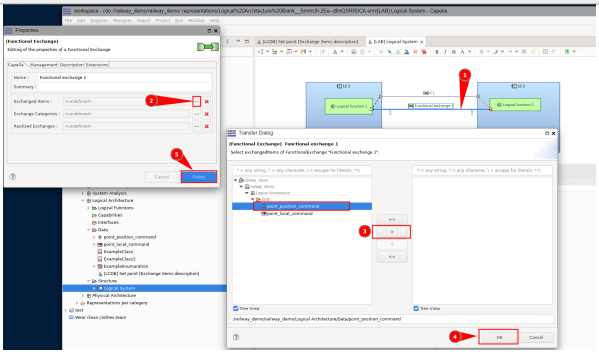
- SPPR-10082 - Function (Logical Function)
- SPPR-10085 - Functional Exchange (Logical Functional Exchange)

Physical Architecture

- SPPR-10082 - Function (Physical Function)
- SPPR-10085 - Functional Exchange (Physical Functional Exchange)

Here is a step by step tutorial to create a Functional flow viewpoint in Capella:

Step description	Illustration
Create in Capella one Data Flow Blank e.g. [SDFB] diagram, to specify functions and functional exchanges To create the data flow diagram 1. Right-click the capability or the Root Function 2. Select New Diagram 3. Select System Data Flow Blank 4. Define the name according to diagram naming convention	
To capture functions: 1. Click-left on System function in the Palette (Function with the symbol if the function creates new function, the blue \ will include a function, that already exists in the model, on the diagram) 2. Click-left on the white space 3. Include name of the function use verb-noun phrase according to Modelling Rules 4. Include a description for the function	
Capture the first defined functional exchange in the model on the diagram.To capture functional exchanges: 1. Click-left on functional exchange in the Palette 2. Click-left on the function that outputs the functional exchange 3. Click-left on the function that has the functional exchange as input 4. Include name of the functional exchange	
Allocate exchange items to functional exchanges 1. Double-click left on the functional exchange either in project explorer SAB, LAB, PAB or SDFB, LDFB, PDFB diagram 2. Click-left on the "..." selection button	

Step description	Illustration
3. Check all the instances of exchange items which are already created. - If one or more of them corresponds to the need, select it/them and allocate it/them to the functional exchange. 4. Before click ok check the following: - If no exchange item corresponds to any functional exchanges, perform the step "Crea te exchange items" step again. Note: Perform this task in parallel to the creation of functional chains and exchange scenarios. 5. Finish	

Method for definition of control loop functions

A general definition for a SPPR-2598 - Function was previously described. This section will further detail how functions can be defined for a railway system of interest.

Hypothesis 1: A railway system of interest is a time-varying, physically distributed, large-scale interconnected multi-input multi-output control system. It involves elements (such as trains, people, signalling system, etc.) that interact with each other in a complex way under multiple types of constraints in order to meet a given need. The overall function can be described as controlling the positions of trains and people on the network in accordance with multiple types of constraints and a complex reference signal (the timetable). This is achieved by using a huge number of diverse sensors and actuators, some of which are inside and some are outside the system border.

Hypothesis 2: Most core functions of a railway system of interest can be assigned to one of the four categories "control", "actuate", "sense" or "observe".

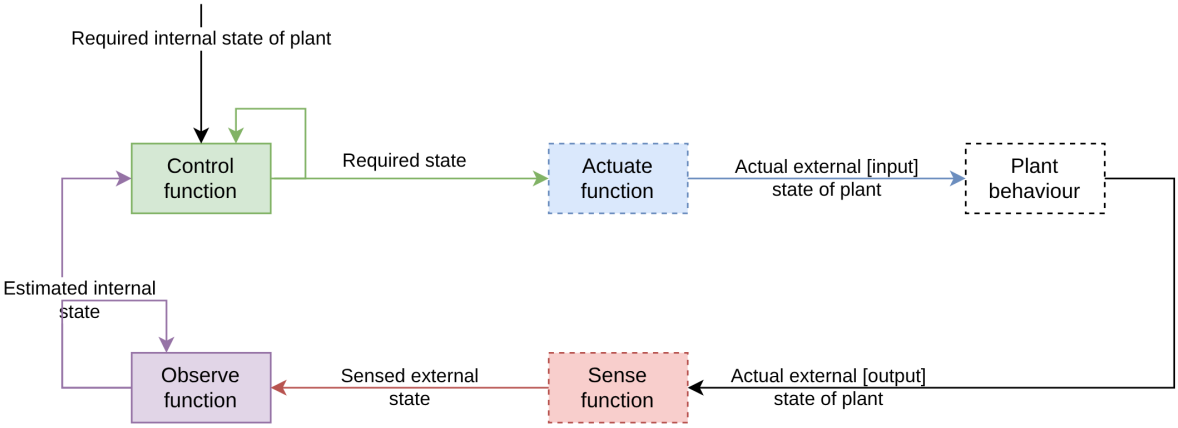
Hypothesis 3: The core functions of the railway system of interest can be modelled as a combination of many of these control loops, which are interconnected and nested inside each other. A railway system contains a collection of several "plant behaviours" that are to be controlled with several control loops. Actor functions belonging to the "Plant behaviour" category represent the physical behaviour of the entity they belong to.

Definition: The "plant" is the system being controlled - in the case of railway system of interest, this is the collection of infrastructure moveable assets, trains, people, and everything else that is controlled by a railway system of interest.

Note: The hypothesis is not that the whole railway system can be reduced to one item of "plant" - the overall system contains a large number of control loops interacting with each other, where each loop controls an associated (internal) plant. This means that there is no single plant, but the entire system consists of multiple (literally a large number of) plants.

Exception 1: Functions associated with non-operational states of railway system of interest, such as data updates or switching between operating and maintenance states, might not fit these categories because they are not part of the chain of functions directly controlling the railway state.

The **main functions of a control system** are illustrated in colour below.



- **Control functions:**
  Makes the decision about a required state change, instruction or command.
  Example: The function "Control point position" controls the position of a point by issuing commands for the point.
- **Actuate functions:**
  Transforms instructions or commands into a physical state.
  Example: The function "Actuate point machine" actuates the point machine of the point.
- **Plant behaviour functions**: see above the definition.
- **Sense functions:**
  Transforms a physical external state into datum.
  Example: The function "Sense position of one point machine" measures the position of a point and forwards the measured information.
- **Observe functions:**
  Makes inferences and/or observations/estimations about the state of the external plant given incoming data and feeding the observations back to the control functions to close the loop.
  Example: The Function "Observed position of one point" translates the sensed position into an abstract value like "no end position".

Note:

- As indicated in the examples above, it is not necessary that all functions of a control loop belong to the system of interest. Some of functions can be allocated to external actors or maybe merged as one function.
- This ontology allows estimated states to be exchanged between observation functions so that observations can be made at different levels of abstraction; in other words, an observer does not always have to transform a sensed external state directly into a fully abstract estimated state;
- Similarly, this ontology allows control functions to produce a required state that is not the required state of an actuator, but rather the required state of some other abstract concept, that feeds to a further control function; in other words, a control function does not always have to transform a required plant internal state directly into a required actuator state, but it might generate a command reference signal for a neighbouring control loop.
- Where a human actor is in a control loop, an additional category of function "indicate" is allowed so that information from observed or other controlled states can be used by the human actor to make their control decisions. But these indicate functions are only needed up to the lowest level of the specification.

Taxonomy for control system principle is explained in modelling rules: ❗ SPPR-3936 - Function name uses recommended verbs

Lessons learned 2025:

- The generic strategy should be to focus more on the interaction between system under consideration and its environment and less about internal functional decomposition and functional exchanges within one system under consideration.

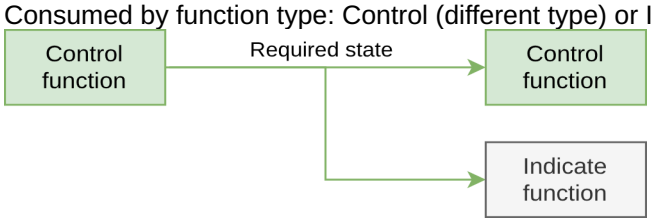
- This strategy should be applied consistently across all levels of analysis. As a result, SP EET propose to avoid internal functional exchanges within the system as much as possible
- Adopting this strategy will enhance the clarity of context diagrams and simplify the overall model, while still allowing us to specify the requirements for the black box system and creating scenarios to show compliments with the stakeholder needs.
- This approach will also help to avoid unnecessary design decision on higher levels, that should be solved on lower levels due to architecture and design decisions
- To implement this strategy, the control loop pattern should still be utilised; however, in certain instances, function categories should be combined into one function. For example, sense and observe should always be merged into a single function, if both function types are assigned to the same system within the current architectural level (in System Architecture, e.g., actor or system; in Logical Architecture, e.g., Traffic Control System or Track Asset Control System, or similar)
- It is important to understand that internal functional exchanges should not be created if the information is available through an external actor. This refers to data or inputs provided by entities outside your system. These inputs are crucial for the system's functionality and should be integrated as inputs to all system functions requiring them.

Possible reasons to include an internal functional exchange could be (not recommended to do it):

- If there exists an external requirement that should already be represented at this architectural level, an internal exchange can be used to represent it. The justification for this should be documented in the exchange's description, e.g., closing the control loop.
- If risk analysis has been conducted, an internal functional exchange might need to be added to represent the safety mitigation in the model. The necessity for the internal exchange should be documented in the description.
- If analysis identifies that multiple functions perform the same task multiple times (e.g., calculate, agree, authorise), then these should be outsourced to a separate function (essentially brought out into a separate section). This is an individual decision that affects the next architectural level and should be rationalised further.

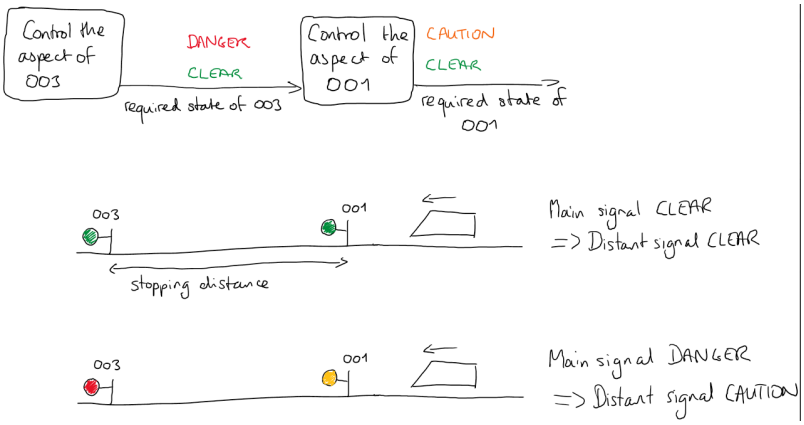
Functional exchanges
For all exchanges between functions that use the control pattern, the following exchange types are permitted:

- **Required state (non-reflexive)**
  - where an abstract required state is used as a parameter to another function
  - Produced by function type: Control OR external input to system
  - Consumed by function type: Control (different type) or Indicate.

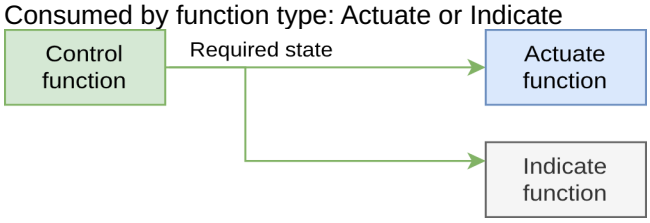


Example (classical route signalling):

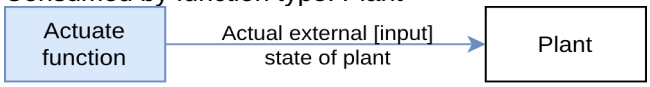
Source	Exchnage	Sink
Control set/unset state of route	Required state of point	Control left/right position of point



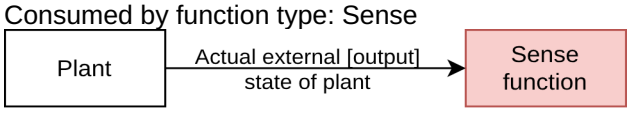
- **Required state (reflexive)**
  - where one type of abstract state is used as a parameter to another instance of the same function that produced the original abstract state type
  - Produced by function type: Control state x (instance y)
  - Consumed by function type: Control state x (instance z)
  - For functional exchanges between two instances of the same function, the functional exchange should go through a duplicate function (Name of the duplicate function shall be Duplicate instance of function 'original function name').
- **Required state**
  - Produced by function type: Control
  - Consumed by function type: Actuate or Indicate



- **Actual external input state**
  - Produced by function type: Actuate
  - Consumed by function type: Plant

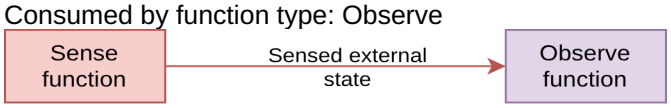


- **Actual external output state**
  - Produced by function type: Plant
  - Consumed by function type: Sense



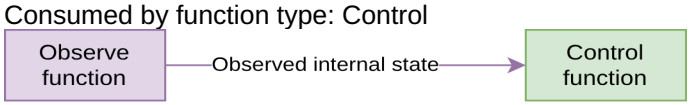
- **Sensed external state**
  - Produced by function type: Sense

◦ Consumed by function type: Observe



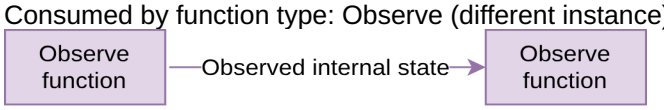
- **Observed internal state (abstraction level equal to abstraction used by control)**

◦ Produced by function type: Observe
◦ Consumed by function type: Control



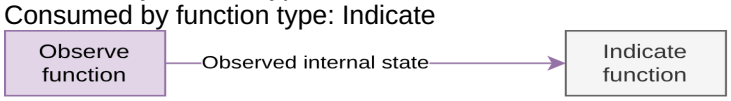
- **Observed internal state (abstraction level less than abstraction used by control OR used by observe function for a different internal state)**

◦ Produced by function type: Observe
◦ Consumed by function type: Observe (different instance)



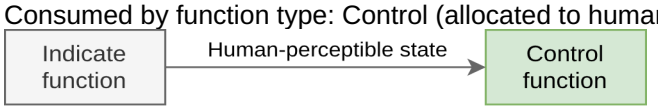
- **Observed internal state (for indication to a human user)**

◦ Produced by function type: Observe
◦ Consumed by function type: Indicate



- **Human perceptable state**

◦ Produced by function type: Indicate
◦ Consumed by function type: Control (allocated to human actor)



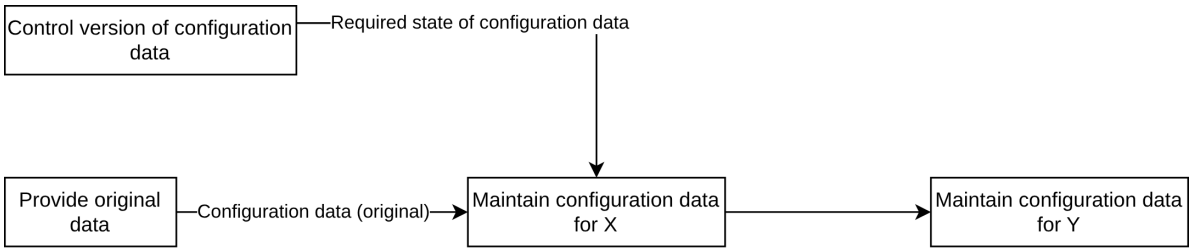
Control loop needs configuration data

Hypothesis 4: The system configuration is another control loop on its own. It is mostly out of sync with the loops of the system itself as the system operations need to be (at least partially) halted in order to apply changes to the configuration.

Hypothesis 5: The creation of the configuration data and their activation as the valid configuration of the system are not synchronised. The planned future configuration is already validated to planning processes of a railway system, while other use cases require data about the valid (i.e. actual) system configuration.

Note: Original configuration data are provided by an external actor (the owner of the data). The provision of data occurs discontinuously, whenever a required state of the system configuration is defined. The activation of the data (the transition from a planned state to the valid actual state) is controlled by an external actor who got the necessary confirmation that the real configuration of the system fits the to-be-activated data in the "Maintain configuration data" function. Example for configuration data: length of a rolling stock: this is a critical parameter for controlling and observing the motion, as well as indicating it accurately.

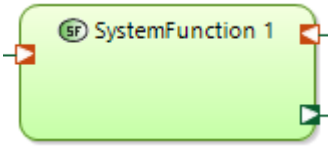
Relation to control loop functions
The configuration data functions does not have a direct functional exchange to control loop functions in order to allow separation of concerns and focus on the data provision and distribution between the systems and subsystems and not on the internal usage of the data within a system.



Interpretation method functions
Functions may appear in different diagrams, such as SPPR-7480 - Interpretation method functional chain diagrams, SPPR-8862 - Interpretation method of exchange scenarios, and SPPR-7264 - Interpretation method context diagram, but they are usually created in an functional data flow diagram. The main concepts are:

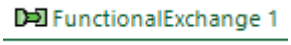
Data Flow Blanks/Diagrams represent the information dependency network between Functions. These diagrams provide a diverse set of mechanisms for managing complexity: simplified links calculated between the high-level Functions, the categorisation of Exchanges, etc.
The Functional Chains can be represented as highlighted paths.

Function

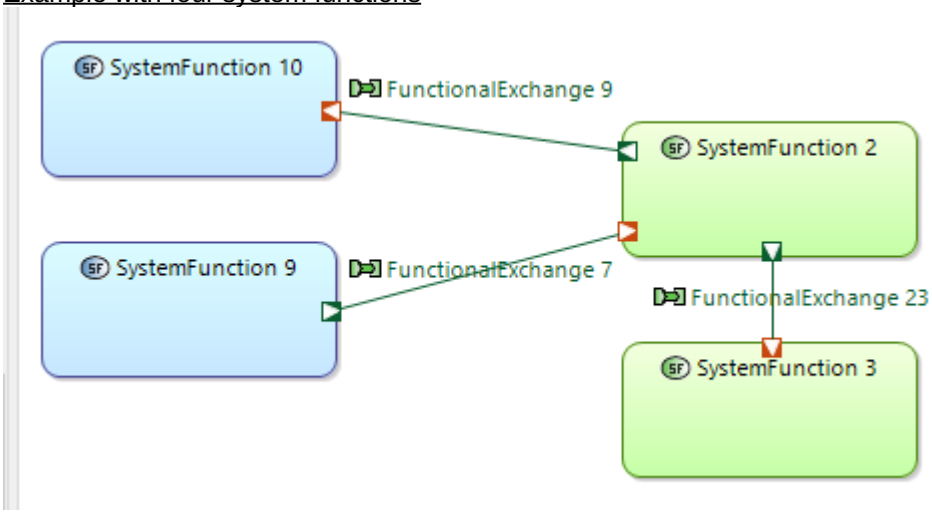


In the diagram, a function may have one or more Function Input Ports ( ) and Function Output Ports ( ), through which it interchange exchange items with other System Functions.

System Functional Exchange



Example with four system functions



3.3.4 View modelling rules

Functions involvement
Describes a consolidated set of system functions and their exchanges which are involved in one system

capability.

1. one diagram exists for each capability
2. at least one function shall be shown
3. at least two actor functions are shown
4. all functions and functional exchanges involved in the fulfilment of the owning capability are shown
5. for functional exchanges on the diagram the associated exchange items shall be displayed, instead of the name of the functional exchange.

Where more than one capability use almost exactly the same set of functions, it is permissible to use one diagram to capture the functions and exchanges of more than one capability. Where a capability involves more functions than can fit sensibly on a diagram, it is permissible to create more than one diagram for the capability. For the functional exchanges it is also alternatively possible to show the functional change name together with the allocated exchange item name.

Notes	[SDFB] [LDFB] [PDFB]
ID	SPPR-6633

Functional flow consolidation
Describes a consolidated set of all defined functions.

1. one diagram exists for each architecture level
2. all defined functions shall be shown
3. all defined functional exchanges shall be shown
4. allocated exchange items on functional exchanges shall be shown instead of the functional exchange names

This is primarily a working view to see and work on every function including their functional exchanges. For the functional exchanges it is also alternatively possible to show the functional change name together with the allocated exchange item name.

Notes	[SDFB] [LDFB] [PDFB]
ID	SPPR-9580

3.4 Function allocation viewpoint

3.4.1 Description, stakeholders and concerns

Description
The function allocation viewpoint is similar to the SPPR-9449 - System context description viewpoint with the difference that allocated functions are shown. For definitions related to the concept of Function allocation, please refer to SPPR-11462 - Description and SPPR-11460 - Description.

Viewpoint stakeholders
The list of stakeholders mentioned in SPPR-10950 - Stakeholders adressed is applicable to this viewpoint.

The specific stakeholders/roles addressed by this viewpoint are:

Creator of this viewpoint :

- SPPR-10696 - Systems Engineer as Modeler : Define and refine functions and allocate them to the <structural entity name>, ensure consistency with the <structural entity name> requirements.

Users of this viewpoint :

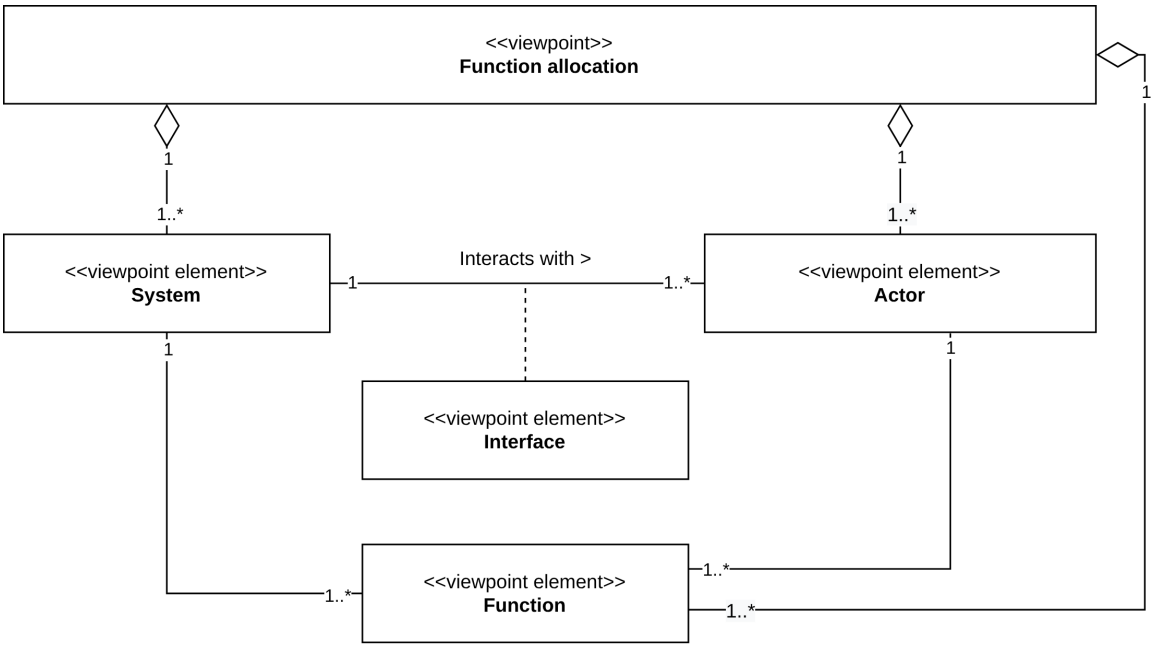
- SPPR-11183 - Test Engineer : Ensure the correctness and completeness of the functions implementation in hardware and software.

Concerns
This list of concerns is not exhaustive and can be completed.

- Are functions correctly distributed across the system or actors?
- Are constraints (e.g. hardware limitations, redundancy, etc.) and non-functional requirements (e.g. performance, reliability, maintainability, etc) considered for the functional allocation?
- Is the level of detail of functions appropriate for the analysis level of the system?
- Is the functional allocation aligned with the system concept and upper level requirements?
- How does the functional allocation affect scalability and future system upgrades?

3.4.2 Visualisation kind

Meta model of the visualisation kind for the function allocation



3.4.3 View method

Construction method function allocation
Capella elements to be used in a function allocation viewpoint are:

System analysis

- SPPR-10060 - Actor (System Actor)
- SPPR-10062 - System (System Component)
- SPPR-10071 - Interface (System Component Exchange)
- SPPR-10082 - Function (System Function)

Logical Architecture

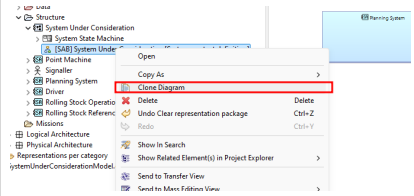
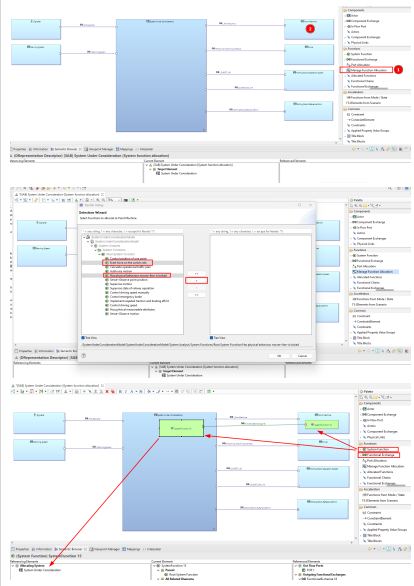
- SPPR-10060 - Actor (Logical Actor)
- SPPR-10062 - System (Logical Component)
- SPPR-10071 - Interface (Logical Component Exchange)
- SPPR-10082 - Function (Logical Function)

Physical Architecture

- SPPR-10060 - Actor (Physical Actor)
- SPPR-10062 - System (Physical Component (Node or Behaviour)

- SPPR-10071 - Interface (Physical Component Exchange)
- SPPR-10082 - Function (Physical Function)

Here is a step by step tutorial to create a Function allocation viewpoint in Capella:

Steps description	Illustration
Allocate functions and exchanges in the model	
Create in Capella one Architecture Blank diagram e.g. [SAB], to allocate functions and functional exchanges. To create the diagram it is also possible to clone the System context definition diagram: 1. Right-click the already created system definition diagram 2. Select Clone Diagram 3. Select System Data Flow Blank 4. Define the name according to diagram naming convention	
Allocate the functions after decision on the alternatives boundaries. To allocate the functions: 1. Left-click on the Manage Function Allocation 2. Left-click on the element where the function should be allocated 3. Select functions that should be allocated and move it to the right field + click ok Alternative functions and functional exchanges can also be directly created on the SAB diagrams.	

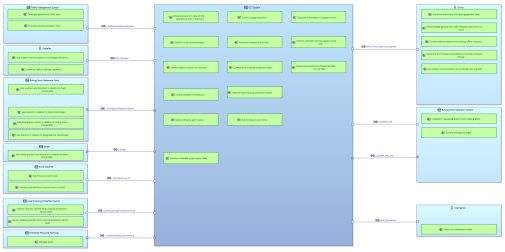
Interpretation method functional allocation
See SPPR-7264 - Interpretation method context diagram and SPPR-7257 - Interpretation method functions.

3.4.4 View modelling rules

Function allocation
Describes the consolidated functional allocations.

1. the system of interest shall be shown
2. all functions shall be shown
3. all actors shall be shown
4. all actor functions shall be shown
5. all component exchanges shall be shown
6. no functional exchanges shall be shown
7. (optional) all system components shall be shown

Notes	[SAB] [LAB] [PAB]
ID	SPPR-6628

Example	

3.5 Interface viewpoint

3.5.1 Description, stakeholders and concerns

Description
The list of definitions related to the concept of Interface is listed below:

Interface
A shared boundary between two systems or between a system and an actor, that interchange exchange items.

Component Exchange
A connection between two components that allows them to convey functional exchange and their exchange items.
In the System Pillar component exchange is used as SPPR-2601 - Interface.

Exchange item
Element transferred through an interface, which can be either energy, matter, and/or information.

A exchange item carries elements with the same transport conditions, simultaneously with the same non-functional properties.

Viewpoint stakeholders
The list of stakeholders mentioned in SPPR-10950 - Stakeholders adressed is applicable to this viewpoint.

The specific stakeholders/roles addressed by this viewpoint are:

Creator of this viewpoint :

- SPPR-10696 - Systems Engineer as Modeler : Define the interfaces of the <structural element name>, align the viewpoint with the <structural element name> architecture and the upper system level interfaces requirements

Users of this viewpoint :

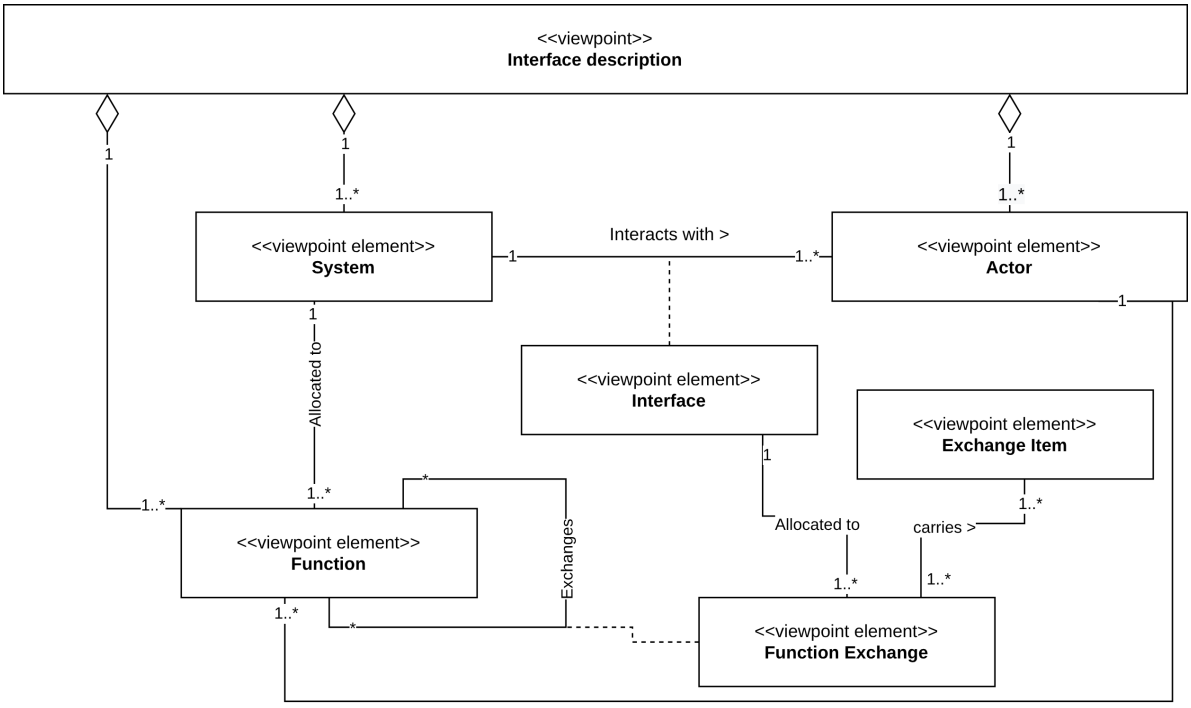
- SPPR-11183 - Test Engineer : Use the interface viewpoint to verify compatibility beteen components and to verify the correctness and completeness of the system interfaces behaviour with its environment.

Concerns
This list of concerns is not exhaustive and can be completed.

- Are all system interfaces (internal and external) and types of exchanges (data, control, material) clearly defined ?
- What communication protocols, means and standards are used ?
- How are errors and failures handled at the interface level?
- Are constraints (e.g. hardware limitations, SW, etc) and non-functional requirements (e.g. performance, reliability, maintainability, security, etc) considered for the interface overview definition ?

3.5.2 Visualisation kind

Visualisation kind meta model Interface description



3.5.3 View method

Construction method interface description
Use Capella Architecture diagrams to model the interface definition. See SPPR-10891 - View modelling rules.

Capella elements to be used in an interface viewpoint are:

System Need Analysis

- SPPR-10060 - Actor (System Actor)
- SPPR-10062 - System (System Component)
- SPPR-10071 - Interface (System Component Exchange)
- SPPR-10082 - Function (System Function)
- SPPR-10085 - Functional Exchange (System Functional Exchange)

Logical Architecture

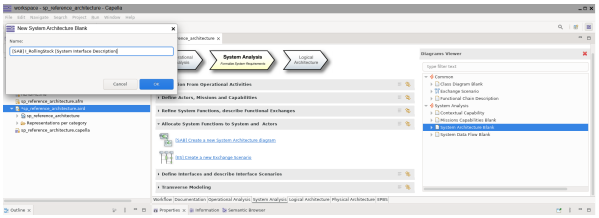
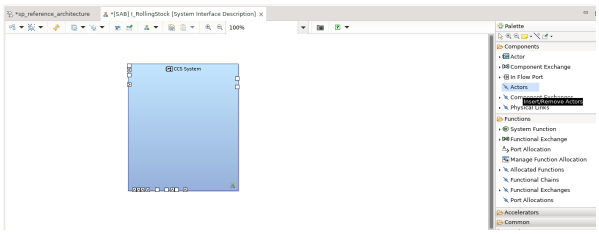
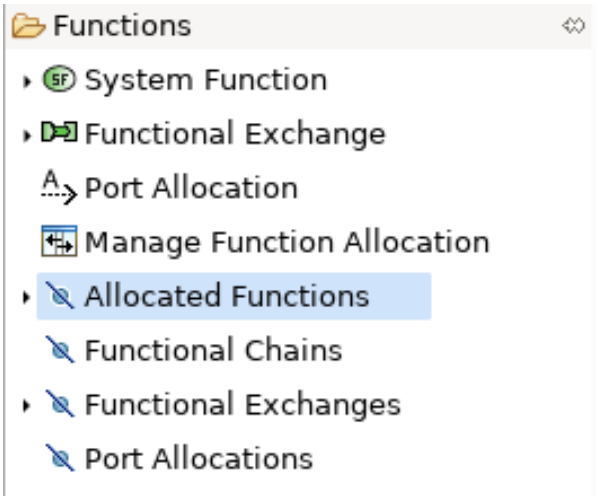
- SPPR-10060 - Actor (Logical Actor)
- SPPR-10062 - System (Logical Component)
- SPPR-10071 - Interface (Logical Component Exchange)
- SPPR-10082 - Function (Logical Function)
- SPPR-10085 - Functional Exchange (Logical Functional Exchange)

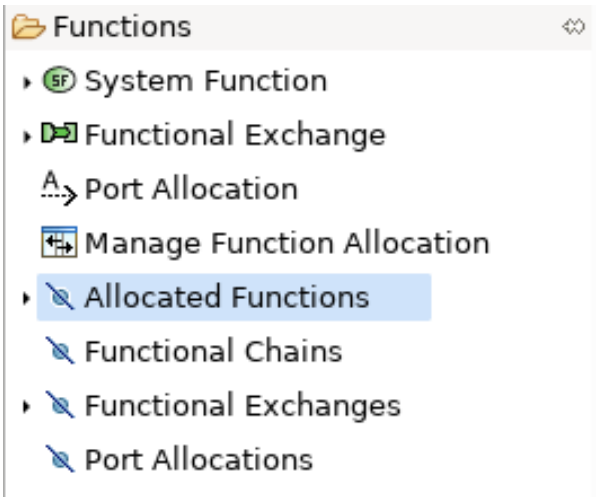
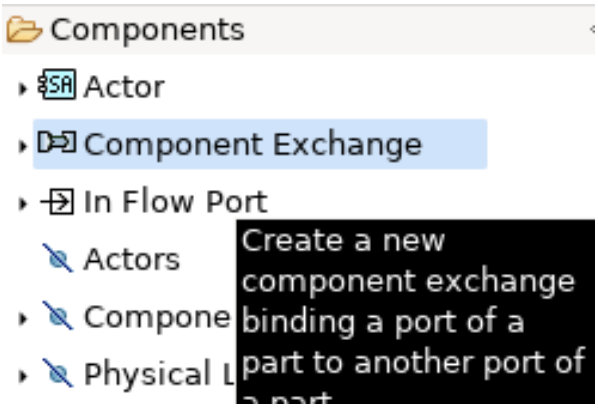
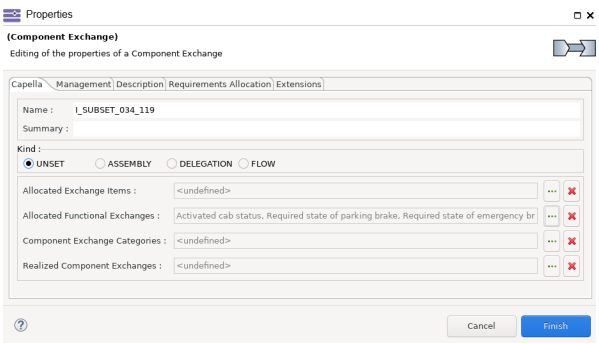
Physical Architecture

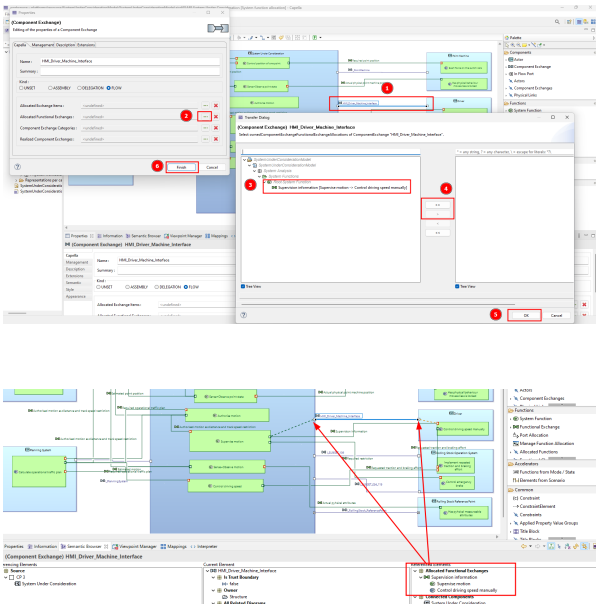
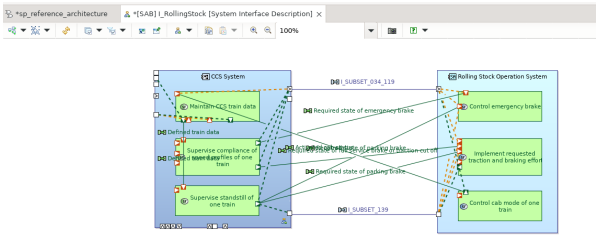
- SPPR-10060 - Actor (Physical Actor)
- SPPR-10062 - System (Physical Component (Node or Behaviour)
- SPPR-10071 - Interface (Physical Component Exchange)

- SPPR-10082 - Function (Physical Function)
- SPPR-10085 - Functional Exchange (Physical Functional Exchange

Here is a step by step tutorial to create an Interface viewpoint in Capella:

Step description	Screenshot
1. Create an Architecture Blank diagram for the Interface Description (in this example we use an SAB for System Need Analysis modelling level) 2. Rename the diagram according to the naming conventions in SEMP annex M2.	
3. Using the Palette, Insert the existing (or create a new) System Actor for which you would like to describe the interface with the system	
If the Interface (Component Exchange) has already been created in another diagram (e.g. System Context description) : 4. Using "Allocated functions" in the Palette and by selecting the System or System actor, display the functions that have functional exchanges allocated to the interface. 5. The functional exchanges will be added automatically to the diagram Go directly to <u>Step 8</u> below.	

Step description	Screenshot
If not : 4. Using "Allocated functions" in the Palette and by selecting the System or System actor, display the functions that have functional exchanges to be allocated to the interface. 5. The functional exchanges will be added automatically to the diagram	
6. Create a new Component Exchange between the System and System actor using the Palette and rename it according to naming conventions	 

Step description	Screenshot
7. Allocate functional exchanges that cross the boundary to component exchanges To allocate the functional exchanges: - Double click-left on the component exchange - Left-click on the ... (Allocated functional exchanges) - Select functional exchanges that should be allocated - Move it to the right field - Left-click ok - Finish After the allocation, this is visible as — line between the functional port and the component port and in addition the allocation is shown in the semantic browser.	
8. Rearrange the diagram to make it readable	

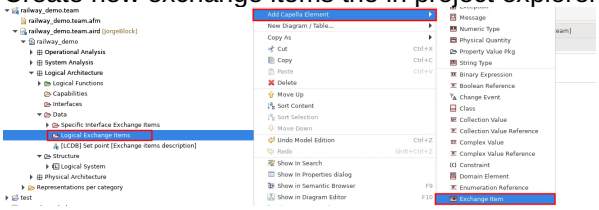
Construction method exchange items

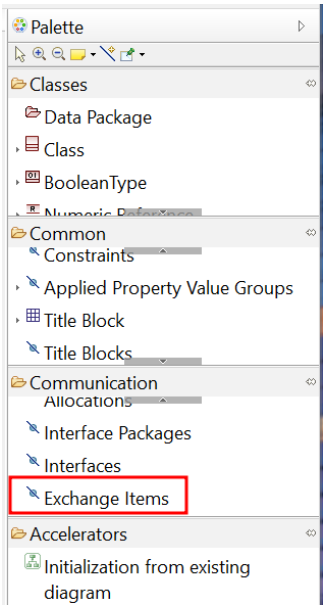
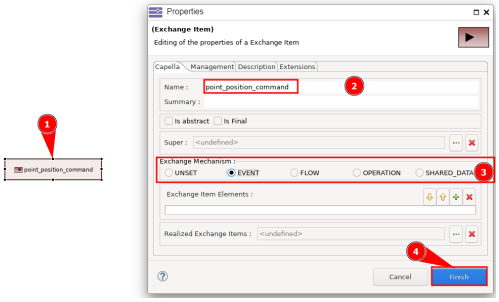
Define the exchange mechanisms and the data that will be transmitted through functional exchanges. Exchange items are carried by a functional exchange, and they help to be more precise which kind of inputs are needed for a function or specific step (e.g. in a scenario).

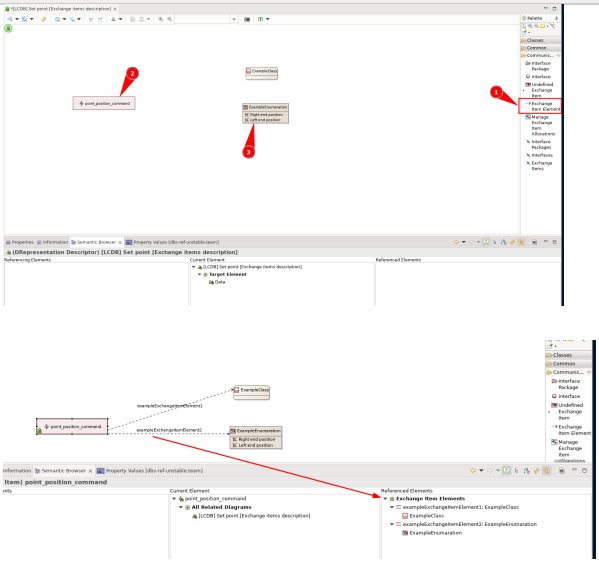
Capella elements to be used in an interface viewpoint are:

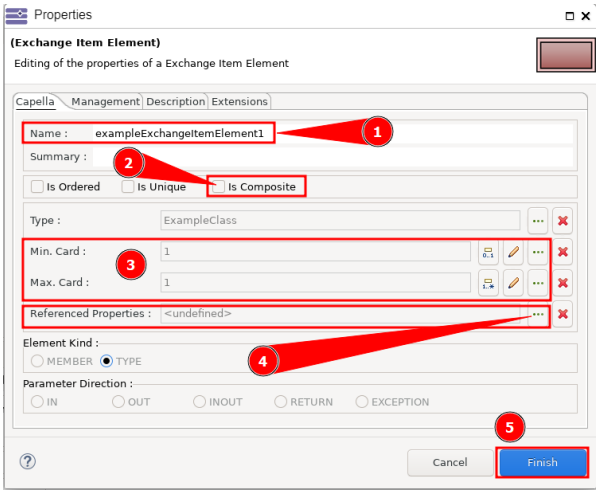
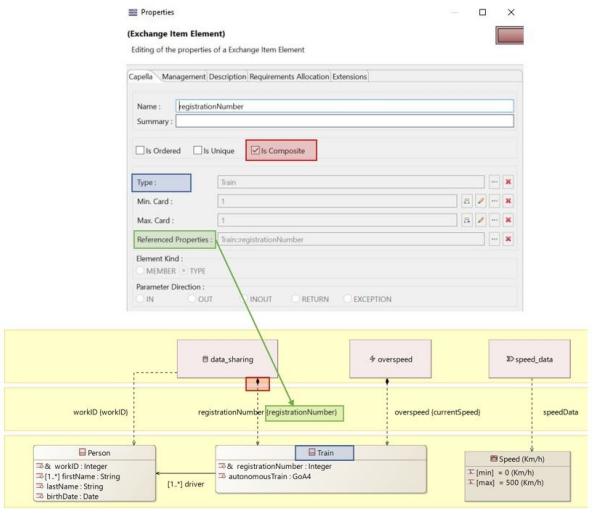
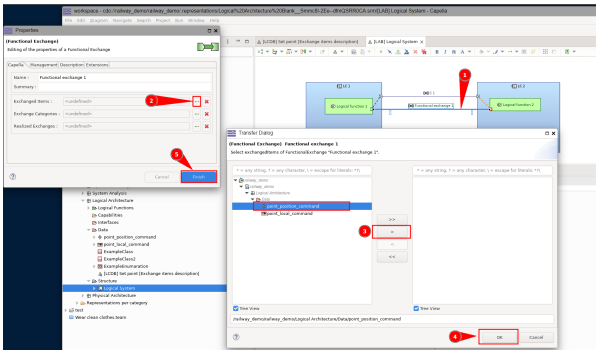
- SPPR-10105 - Exchange Item
- SPPR-10104 - Class

Here is a step by step tutorial to create Exchange items in Capella:

Description	Illustration
Create exchange items directly in the project model in the dedicated data packages. (If the exchange items do exist already then go to the next step).	Create new exchange items the in project explorer:  Create a new package by right-click on the data

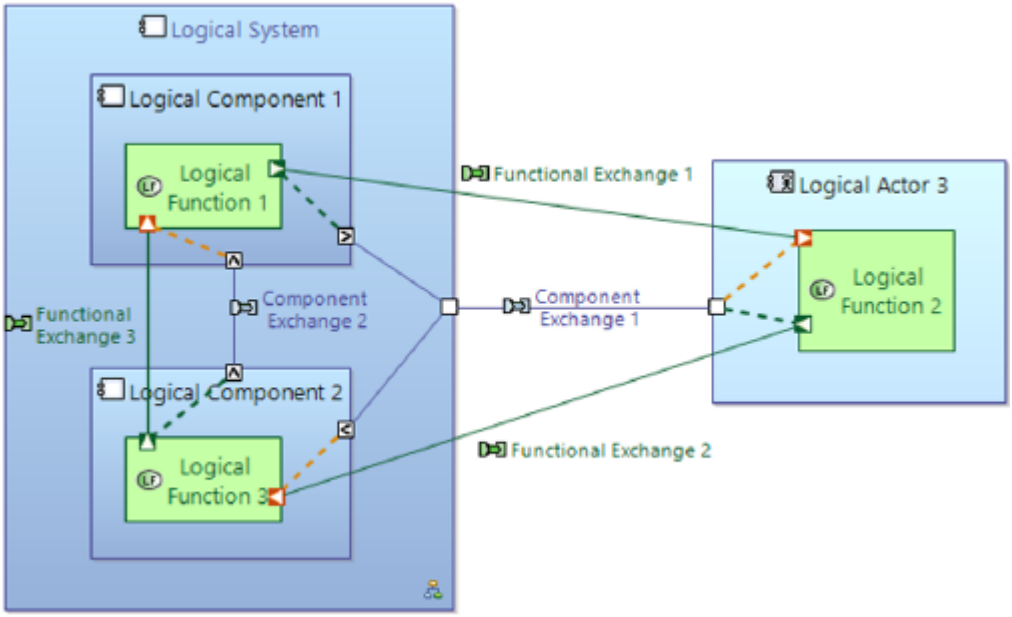
Description	Illustration
Note: • on each modelling layer there exists a data package called "[modelling layer] Exchange Items". • There could be also exist already defined data packages (e.g. per interface).	package: 
Create the exchange item diagram (If the diagram does exist already go to the next step). 1. Right-click the exchange item package 2. Select "New Diagram" 3. Select "Class Diagram Blank" 4. Define the name according to diagram naming conventions defined by the modelling rules ❗ SPPR-8177 - Exchange items description	
Display exchange items and data object classes on an exchange item diagram (If the exchange items do exist already then go to the next step). Note: For each capability (at any modelling layer) an exchange item diagram is defined. The exchange items are added to the corresponding diagram according to their relevance to the capabilities.	
Detail the exchange item information 1. Double left-click on the exchange item 2. Refine the name in line with modelling rule ❗ SPPR-4195 - Exchange item has an unique name 3. Select an exchange mechanism (flow, event or unset mechanisms) for each exchange item ❗ SPPR-4196 - Exchange item has an exchange mechanism 4. Finish	
Create exchange item elements between data object classes (or simple type or structured type)	

Description	Illustration
and exchange items in the exchange item diagram. 1. Left-click on the exchange item element in the palette 2. Left-click on the exchange item 3. Left-click on the data object classes, simple type or structured type	

Description	Illustration
Define exchange item element details 1. Refine the name of the according to modelling rules 2. By default, the exchange item element created has a composition relationship between the data object class and the exchange item. Depending on whether the data object class is specific or not to this exchange item, the composition characteristic should be removed. - in case of a class is specific only to one exchange item then the exchange item element is composite - in case of a class is used for multiple exchange items then it is not is composite 3. Define the cardinality of the created exchange item elements [1..n] 4. It is also possible to refer to specific properties of a data object class 5. Click Finish	 
Allocate exchange items to functional exchanges 1. Double-click left on the functional exchange either in project explorer SAB, LAB, PAB or SDFB, LDFB, PDFB diagram 2. Click-left on the "..." selection button 3. Check all the instances of exchange items which are already created. - If one or more of them corresponds to the need, select it/them and allocate it/them to the functional exchange. 4. Before click ok check the following: - If no exchange item corresponds to any functional exchanges, perform the step "Cre ate exchange items" step again. Note: Perform this task in parallel to the creation of functional chains and exchange scenarios. 5. Finish	

Interpretation method interface and exchange item
An Interface viewpoint is particularly useful to define and represent **internal interfaces** between components within a system and **external interfaces** between the system and external actors. It helps in visualizing the **data exchanges, service dependencies,** and **functional interactions** between the different entities.

Example of an Interface description diagram (Logical Architecture level):



In this example, we can first of all see the **structural elements** (blue rectangles), i.e. the Logical Components (contained in overarching box that represents the System at the Logical level) and the Actors.

Next, we can see the **Functions** (green rectangles), within the Logical Components or the Actors : this is the allocation relation. One or several Functions can be allocated to the same structural element. This is the case for Functions 2, which is allocated to Actor 3, as well as for Functions 1 and 3, which are allocated to the System.

The **Functional Exchanges** (green arrows) are represented between the Functions, always linking a Function Port from a Function output (green square) to a Function Port from the input of another Function (orange square).

One or more Functional Exchanges can be allocated to the same **Component Exchange** (blue line). This is the case for Functional Exchanges 1 and 2, which are both allocated to the Component Exchange 1, as shown by the dotted line linking the Function Ports to the Component Ports.

A Component Exchange either links the Logical System to one of its Actors, or a Logical Component directly to an external Actor, or two Logical Components via Component Ports (uni- or bidirectional; white squares).

In this example, the Ports of Component Exchange 2 that link the two Logical Components inside the System are unidirectional, since only one Functional Exchange is allocated to it.
On the other hand, Component Exchange 1, which still links Actor 3 to the System, is bidirectional since the Functional Exchanges are of opposite directions. Nevertheless, it is delegated to the two Logical Components via unidirectional Component Ports, each belonging to a different Logical Component.

Exchange item mechanisms
The exchange mechanisms can be of 3 kinds:

- FLOW: It should be used to exchange items by default because data information between two functions is permanently present

- EVENT: It is considered as an asynchronous mechanism, structured data sent from one component or function and received by a unique receiver (unicast), a defined set of receivers (multicast) or an undefined set of receivers (broadcast).
- UNSET: When the information content is unknown or unclear. Add a text description on the exchange item, to act as a placeholder/STUB until the information content is resolved.

The information content of the exchange item should be defined as far as possible using an appropriate combination of data objects (simple types or structured types) thanks to exchange item elements.

Exchange Item Elements characteristics
Exchange items are structured through exchange item elements in the same way as classes are structured in properties: they define the types of data by referencing a data model element (class or structured/simple types).

Exchange item elements have several characteristics:

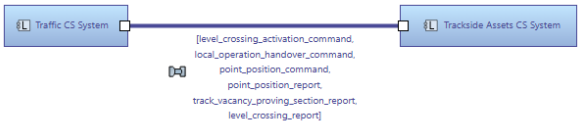
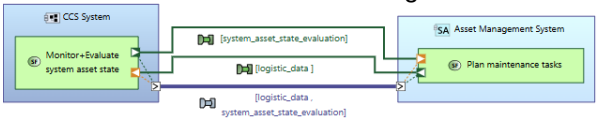
- Composite: specify that the exchange item is a container for the object contained in the element. In other words, where the data object is NOT specific to that exchange item, the "Is Composite" box is UNCHECKED. (represented by a black diamond)
- Type: specify which data model element (classes, data types) is involved in the exchange item
- Cardinality
- Referenced properties: in case of structured type or class we can select specific properties for the exchange item element. (represented with brackets)

3.5.4 View modelling rules

Interface description
Describes one interface of the system to one considered system actor or between two systems.

1. one diagram exists for each interface
2. both interface partner shall be shown
3. only component exchanges between the interface partner shall be shown
4. (optional) only functional exchanges between the interface partner shall be shown

Filter settings are set to display the exchange items that are exchanged over each functional exchange.

Notes	[SAB] [LAB] [PAB]
ID	SPPR-6626
Example	Interface view without functions auto generated from C2P:  Interface view with functions auto generated from C2P: 

Exchange items description
Describes the data items which are exchanged between functions.

1. at least one diagram exists per interface
2. all exchange items shall be shown which are allocated to functional exchanges

3. all data objects of each exchange item are shown
4. all exchange item elements are shown

Data objects that itself does not have a exchange item element link to an exchange item, but are the type for an attribute are shown as well.

Notes	Diagram type abbreviation: [SCDB] [LCDB][PCDB]
ID	SPPR-8177
Example	See construction method.

3.6 Exchange scenario viewpoint

3.6.1 Description, stakeholders and concerns

Description
The list of definitions related to the concept of scenarios is listed below:

Exchange scenario
Describe the interactions between structural elements by focusing on the exchange of information in a given context and with a time axis. They allow the ordering of information sequences and behaviours of structural elements, but can also be used as a basis for specification tests. Scenarios can be linked to sequence diagrams (SysML wording).

Scenarios are suitable to be used for:

- Regarding abstractions: operational needs, system analysis, logical architecture, physical architecture
- Regarding structural entities: system, subsystem, system element, logical component, actor
- Regarding functions: behaviour definition of function (how functions of structural entities exchange data through exchange items)
- Regarding other behavioural aspects: pre and post-conditions and invariant as start to end conditions as well as state invariant of the scenario
- Regarding purpose: represent, at least, one complete sequence of functional exchanges on a time axis

The goal is to visualise system black-box behaviour or system interaction behaviour with actors.

Viewpoint stakeholders
The list of stakeholders mentioned in SPPR-10950 - Stakeholders adressed is applicable to this viewpoint.

The specific stakeholders/roles addressed by this viewpoint are:

Creator of this viewpoint:

- SPPR-10696 - Systems Engineer as Modeler : Align this viewpoint with the system architecture and define its related requirements.

Users of this viewpoint:

- SPPR-11183 - Test Engineer : Use the scenario to verify the correctness and completeness of the system behaviour.

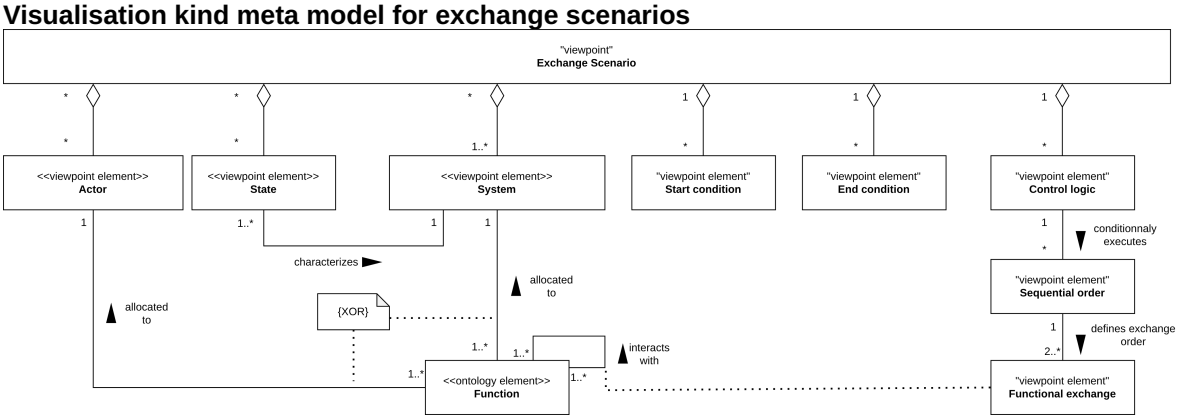
Concerns
This list of concerns is not exhaustive and can be completed.

- Which systems or actors participate in the scenario and inputs/outputs are exchanged between them?
- Are all dependencies for exchanges (data, formats, timing) properly defined?
- What are potential failure points or delays in the exchange?

- Is the boundary of the scenario diagram correctly defined (related capability, systems and actors involved in the capability)?
- Is the level of detail of the scenario appropriate for the analysis level?

3.6.2 Visualisation kind

Visualisation kind meta model for exchange scenarios



3.6.3 View methods

Construction method exchange scenario

In general SPPR-2066 - Exchange scenarios are used in each level of the model, but the possibilities of representation are similar. The process of creating the Exchange Scenario diagram is intended to reveal missing or superfluous functional exchanges or weaknesses in decomposition. No model element should be created or added in a scenario. The Exchange Scenario diagram only displays model elements that already exist.

Note: The intended meaning is that, at a point of the sequence, the function's output changes in line with changes to its inputs. It is NOT intended to indicate that the function is "called" or "invoked"; the function is assumed to be "always-on in a particular system state".

Use Capella Scenario Diagrams to model Exchange scenarios. See SPPR-10800 - View modelling rules. Capella elements to be used in an exchange scenario viewpoint are:

All layers

- SPPR-10105 - Exchange Item
- SPPR-10110 - State

Operational Analysis

- SPPR-10221 - Operational Entity
- SPPR-10222 - Operational Interaction
- SPPR-10220 - Operational Activity

Systen Need Analysis

- SPPR-10060 - Actor (System Actor)
- SPPR-10062 - System (System Component)
- SPPR-10082 - Function (System Function)

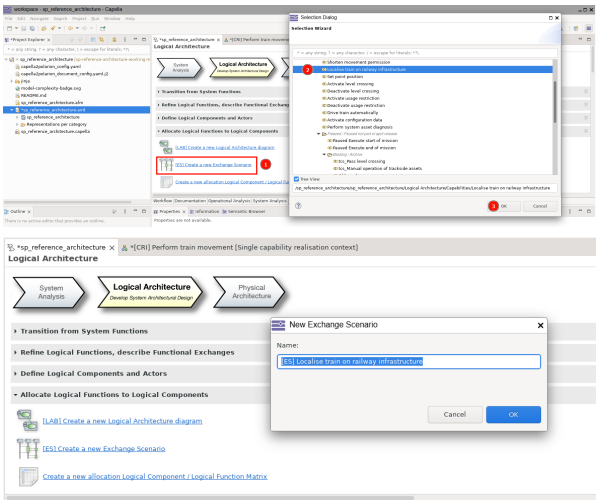
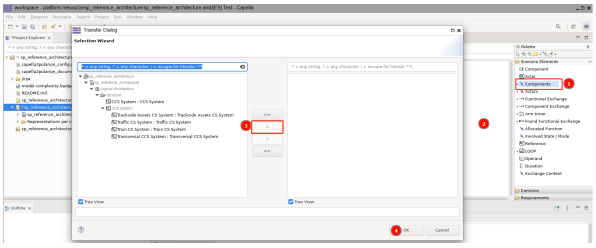
Logical Architecture

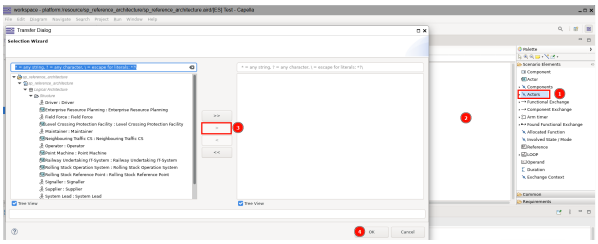
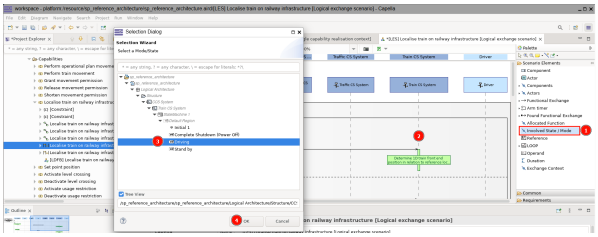
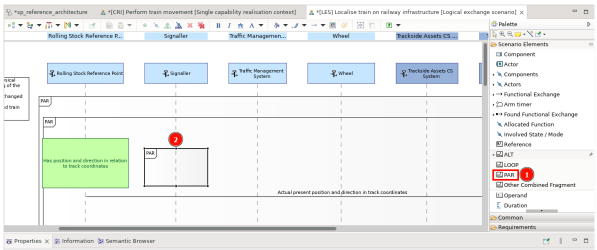
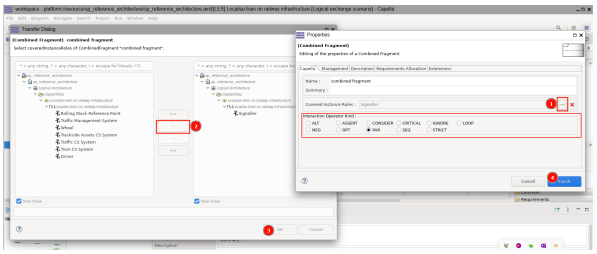
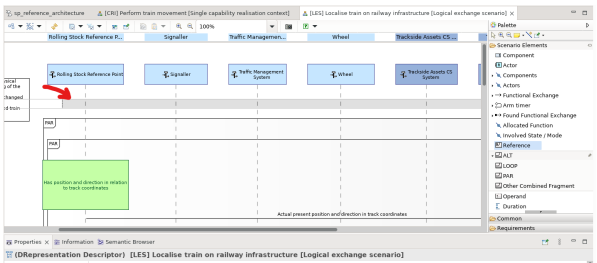
- SPPR-10060 - Actor (Logical Actor)
- SPPR-10062 - System (Logical Component)
- SPPR-10082 - Function (System Function)

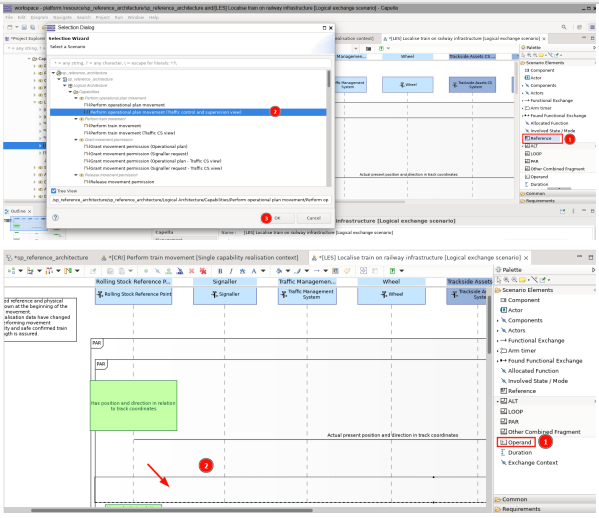
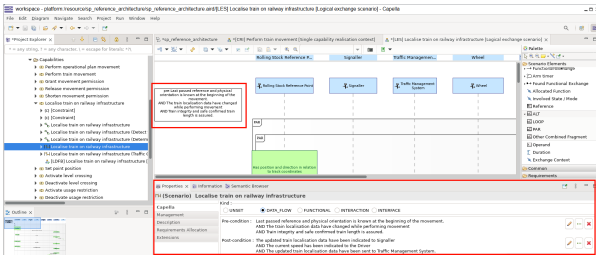
Physical Architecture

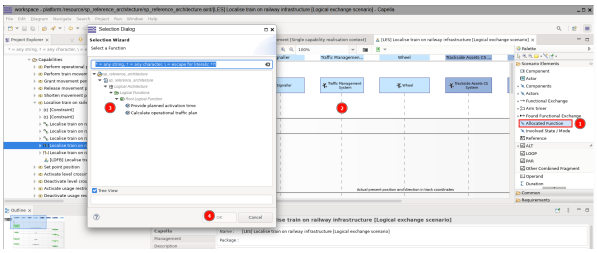
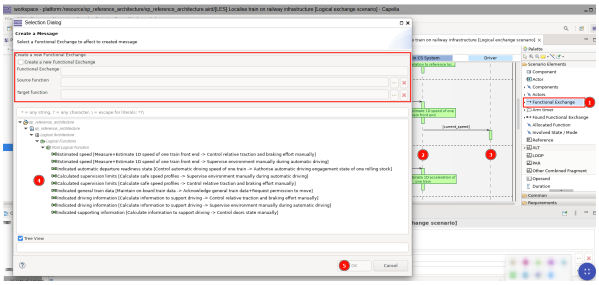
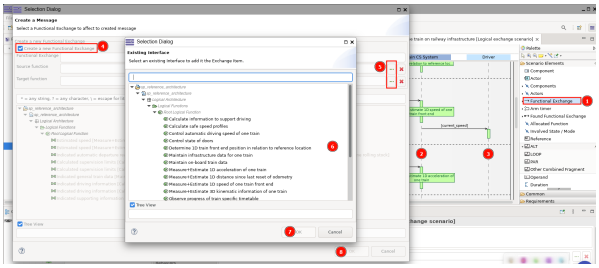
- SPPR-10060 - Actor (Physical Actor)
- SPPR-10062 - System (Physical Component Node or Behaviour)
- SPPR-10082 - Function (Physical Function)

Here is a step by step tutorial to create an Exchange scenario viewpoint in Capella:

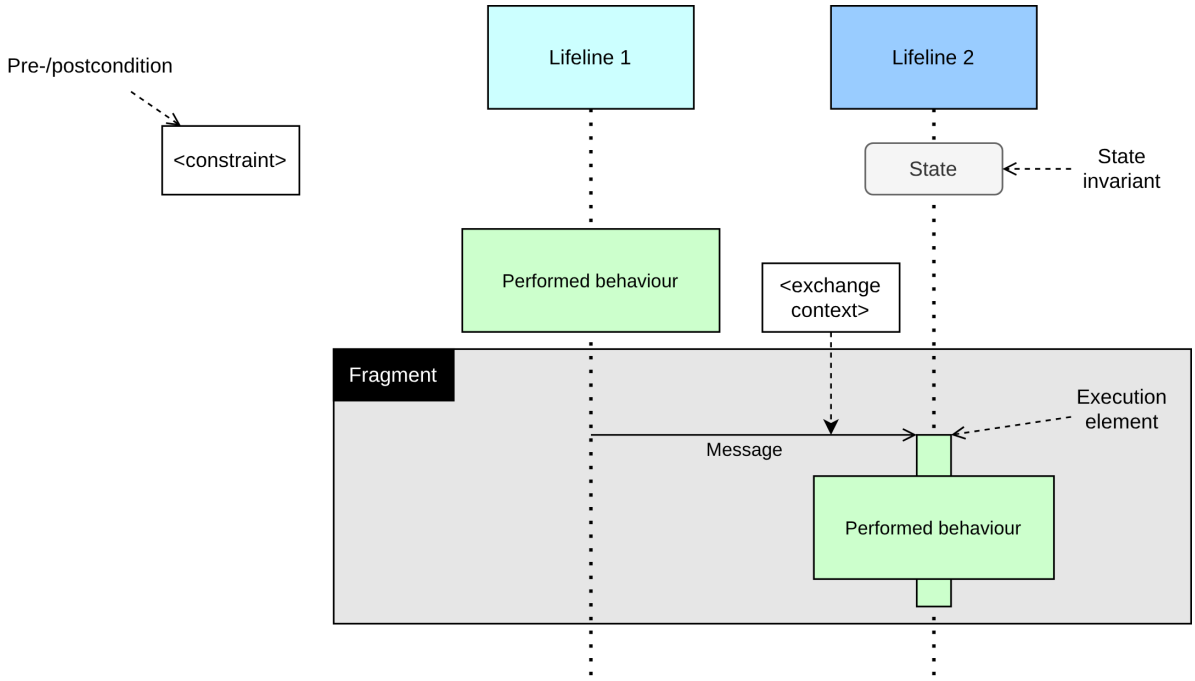
Step description	Illustration
1. Create an Exchange Scenario diagram for the viewpoint (in this example we use an LES for Logical Architecture modelling level) 2. Choose for which capability you would like to create the Exchange scenario 2. Rename the diagram according to the naming conventions in SEMP annex M2.	
3. Using the Palette, Insert the existing logical components involved in the Exchange scenario **Existing components** To add the existing logical components : - Left-click on "Components" - Left-click in the white area of the diagram - Select the components that should be added to the viewpoint - Move it to the right field - Left-click ok Nota : On System Need Analysis modelling level, the system will be added automatically to the Exchange scenario diagram (step 3 is to be ignored)	

Step description	Illustration
4. Using the Palette, Insert the existing Actors involved in the Exchange scenario **Existing actors** To add the existing logical actors : - Left-click on "Actors" - Left-click in the white area of the diagram - Select the actors that should be added to the viewpoint - Move it to the right field - Left-click ok	
Adding Modes & States Using the Palette, Insert an existing mode/state of the logical component To add an existing mode/state : - Left-click on "Involved State / Mode" - Left-click on the logical component's lifeline - Select the desired Mode / State - Left-click ok	
Adding Control Sequence Elements Using the Palette, Insert a control sequence element type (ex. PAR) To add a control sequence element : - Left-click on control sequence element (ex. PAR) - Left-click on the actor's or logical component's lifeline Nota : By double clicking on the control sequence element (ex. PAR), you have the possibility to modify its properties. For example to : -Change its type -Add other instance roles to cover -Add a description To add a control sequence element of type Ref (Reference to another Exchange Scenario) : - Left-click on control sequence element "Ref"	  

Step description	Illustration
- In the diagram area where you want to refer the Exchange scenario, Hold the left-click and drag to cover the involved instances - Select the desired Exchange scenario to reference - Left-click ok To add an Operand (new field) to the control sequence element : - Left-click on "Operand" in the palette - Left-click inside the control sequence element	
Adding Pre- and Post- conditions By filling the Pre- and Post conditions' fields in the Exchange scenario properties, they will be added automatically to your diagram in form of a constraint. Rearrange it to make the Pre-condition in the Top left of the diagram and the Post-condition in bottom-left.	

Step description	Illustration
5. Using the Palette, Insert the existing allocated functions involved in the Exchange scenario To add an existing function: - Left-click on "Allocated Function" - Left-click on the actor's or logical component's lifeline - Select the desired function - Left-click ok	
6. Using the Palette, Insert an existing (or create new) functional exchange involved in the Exchange scenario **Existing Functional Exchange** To add an existing functional exchange: - Left-click on "Functional Exchange" - Left-click on the 1st actor's or logical component's lifeline to define the source of the Functional Exchange - Left-click on the 2nd actor's or logical component's lifeline to define the target of the Functional Exchange - Select the corresponding functional exchange - Left-click ok **New Functional Exchange** To add a new functional exchange: - Left-click on "Functional Exchange" - Left-click on the 1st actor's or logical component's lifeline to define the source of the Functional Exchange - Left-click on the 2nd actor's or logical component's lifeline to define the target of the Functional Exchange - Check the functionality "Crate a new functional exchange" - Fill in the fields "Functional Exchange" (name), "Source Function" and "Target Function". - Left-click ok	 

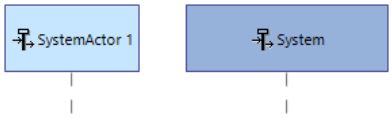
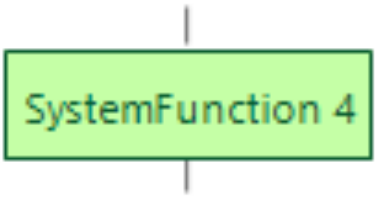
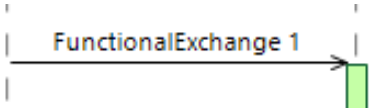
Interpretation method of exchange scenarios
The following picture illustrates the main elements and concepts used in exchange scenario diagrams. A scenario describe how the lifelines interact with a vertical axis representing time.

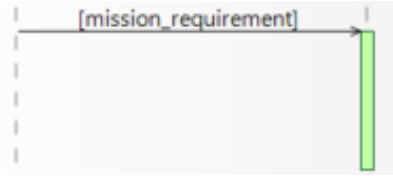


Their constitution, from top to bottom and from right to left, makes it easier to understand the elements described, even for people unfamiliar with an MBSE approach.

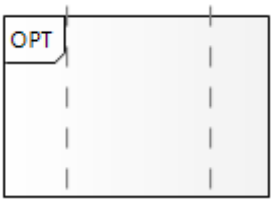
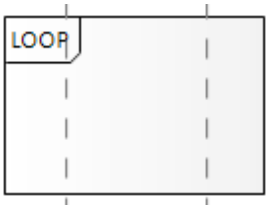
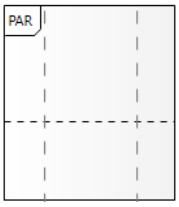
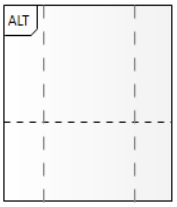
General elements used in exchange scenarios in Capella

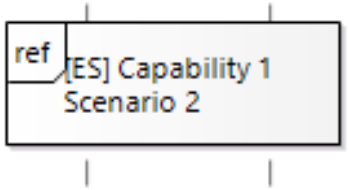
The following main elements are used in scenarios:

Modelling Notation	Element Name	Meaning
	Lifeline	It is the representation of an existence of a model element that participates in the scenario. It has the same name as the model element referenced and is represented graphically with a dotted vertical line. In the case of exchange scenarios, lifelines are the system of interest/system actors/components.
	Performed behaviour	Blocks displayed in a scenario on a lifeline are called "performed behaviours" and represent functions (or activities). They can only be created to represent functions that already exist, so it is helpful to have other diagrams opened where you can create new functions (or activities) and functional exchanges if you need them. When they are deleted, they do not delete the element they represent. They are placed on a lifeline preceding incoming and/or outgoing messages and their outputs change in line with changes to their inputs. Note: The representation of a function as a performed behaviour is done only if it had value for the reader.
	Messages	It is a unidirectional exchanged item between lifelines that triggers a behaviour in the receiver. In the case of exchange scenarios, messages are functional exchanges. Sequence Messages are depicted as originating from a Source Lifeline to a Target Lifeline, although in fact

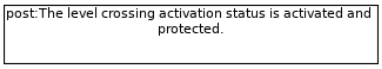
Modelling Notation	Element Name	Meaning
		they represent exchanges between two Allocated Functions.
	Execution element	This can be a label with a shape of a rectangle notating the time in which a performed behaviour is actually active. This is the case for example for an incoming message as long as the duration in which the outputs of a performed behaviour can change within a lifeline. This means as long as new messages with a changed value to another lifeline are happening as depicted in the scenario

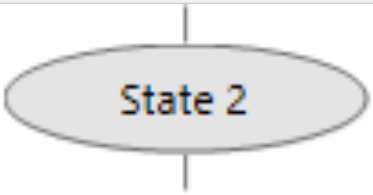
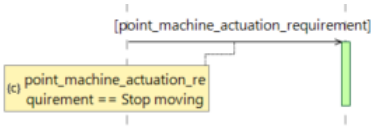
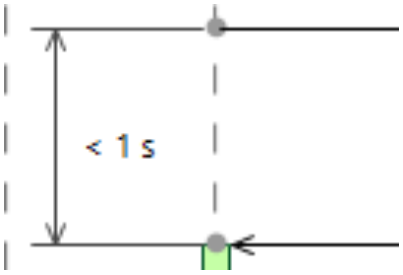
Control sequence elements of exchange scenarios
In addition to the fundamental elements of a exchange scenario listed in the table above, combined fragments can be defined in addition as areas within the scenario, in order to express different cases and changes. A portion of a scenario affected with a specific operator that qualifies its execution in time, and one or more operands (sub-portions of the Combined Fragment). The most used operators are:

Modelling Notation	Element Name	Meaning
	OPT	Used when a message takes place under certain conditions with no alternative. It is a representation of the common "if". The fragment is executed only if the provided condition is true.
	LOOP	Used when a sequence needs to have iterations under certain conditions. The fragment is executed a number of times, depending if the guard condition is true.
	PAR	Used if at least two messages can occur in parallel which means that there is no implied order between occurrences in the different operands.
	ALT	Usually, this possibility should not be needed since alternative paths are usually modelled in separate scenarios. But in several cases, it can represent the "if-then-else" construction of messages, under certain conditions (which must be mutually exclusive) with alternative messages. Note: Where possible, guard conditions should be expressed in terms of the states of entities, actors.
	REF	Used to reduce the complexity of a scenario within one capability, it should be well justified and really only used where it makes sense. Also, the reference object must be laid on the diagram in such a way those non-

Modelling Notation	Element Name	Meaning
		involved lifelines are not covered by the reference object. A reference to an existing scenario, used as part of the execution of another scenario. If a capability includes another capability with the "include"-relationship, then the scenarios of the included one are referenced using the REF combined fragment. No lifelines in the scenarios are shown that do not have any interaction with another lifeline. If another scenario is referenced in which the omitted lifeline is involved, its involvement is only shown in the referenced scenario, but not in the referencing scenario. This differs from the usual practice (e.g. SysML) of displaying all lifelines involved in the capability (i.e. the including and the included capability). If a capability extends to another capability with the "extent"-relationship, then the scenarios of the extended one are referenced using the REF combined fragment and OPT combined fragment. The OPT fragment includes the condition when the extension is applicable. No lifelines in the scenarios are shown that do not have any interaction which another lifeline. If another scenario is referenced in which the omitted lifeline is involved, its involvement is only shown in the referenced scenario, but not in the referencing scenario. This differs from the usual practice (e.g. SysML) of displaying all lifelines involved in the capability (i.e. the including and the included capability).

Specific elements of exchange scenarios
The following specific elements serve as additions to the main elements.

Modelling Notation	Element Name	Meaning
	Pre- and post-conditions	In case the capability owning the scenario has pre and post conditions, and if they are fully applicable to the scenario, they should be displayed on the scenario as well, in order to refine or have more concrete and specific constraints. In some cases the conditions in the owning capability are not fully applicable to the scenario. If they appear in this case, it must be mentioned in the scenario in order to inform the users.
	Invariants for states	Statements about modes and states (called "state invariants") can be displayed in a scenario to express the current state of a lifeline or a change as a result of an interaction. Invariants express some conditions of the lifeline at a certain point in time of the scenario.

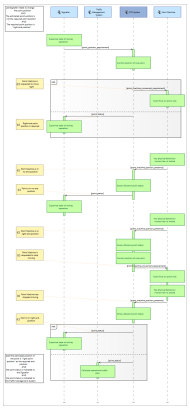
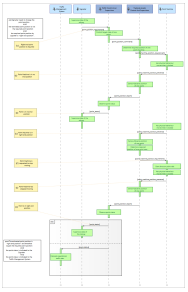
Modelling Notation	Element Name	Meaning
		Depending on the tool, a state invariant can be shown on the lifeline referring to a state of a state machine (e.g. of the structural entity). State changes of entities/ actors (i.e. by their state machines) should be visualised on the lifelines using the provided tool, to aid understanding; this is not mandatory but is helpful to the reader. Be careful, the trigger of the state transition should be consistent with the interaction depicted in the scenario. Note: if there are no state or specific conditions in the scenario, it is assumed that the function is "always-on" or at least "always-on in a particular system state" An Involved State of the System or a System Actor, which characterises its context and define its behaviour. It may appear in a Scenario to illustrate the instant in which a transition to that state takes place and, thus, enhance the understanding of the system's behaviour.
	Exchange context	It is used to illustrate the content of a message and it is displayed as a constraint linked to the message.
	Duration Constraint	A constraint set for the elapsed time between two points of the scenario.

3.6.4 View modelling rules

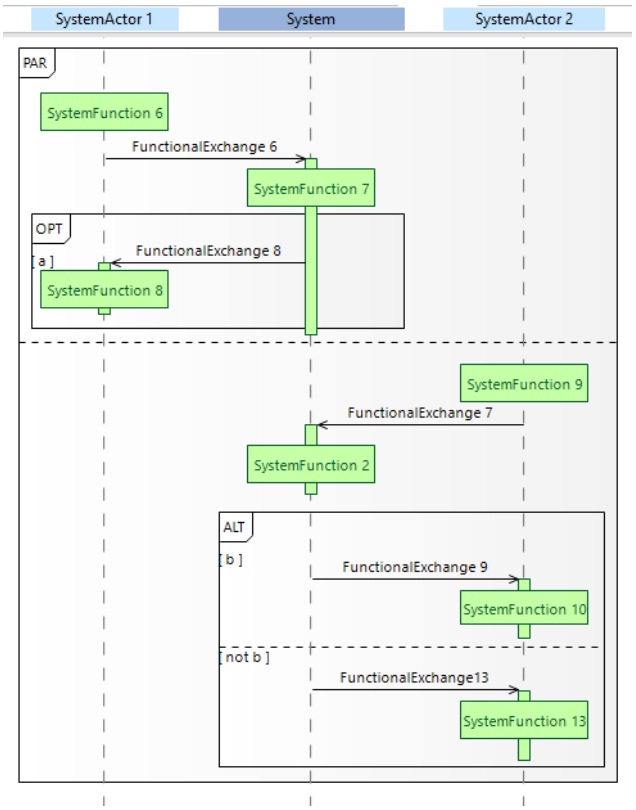
Exchange scenario

1. at least one diagram exists for each capability
2. the system of interest shall be shown as a lifeline
3. at least one actor shall be shown as a lifeline
4. all actors shall be shown always in the same order
    a. the main system under consideration in the middle
    b. the actors at the borders of the systems
    c. the actors and systems sharing a direct interface together; e.g. driver-onboard CCS or Signaller-Trackside CCS.
5. only functional exchanges that leave the boundary of a lifeline are shown
6. internal exchanges are not shown

7. allocated exchange items on functional exchanges are shown instead of the functional exchange name
    a. Specific exchange item is selected, in case that there are more exchange items allocated to this functional exchange.
8. pre- and postconditions of the exchange scenario shall be specified in the related field in Capella, representing a refinement or more concrete conditions that apply to this specific scenario
9. Exchange contexts should be used to indicate additional information for a functional exchange/ exchange item.
10. Notes are not used

Notes	[SES] [LES] [PES]
ID	SPPR-8176
Example	System exchange scenario:  Logical exchange scenario: 

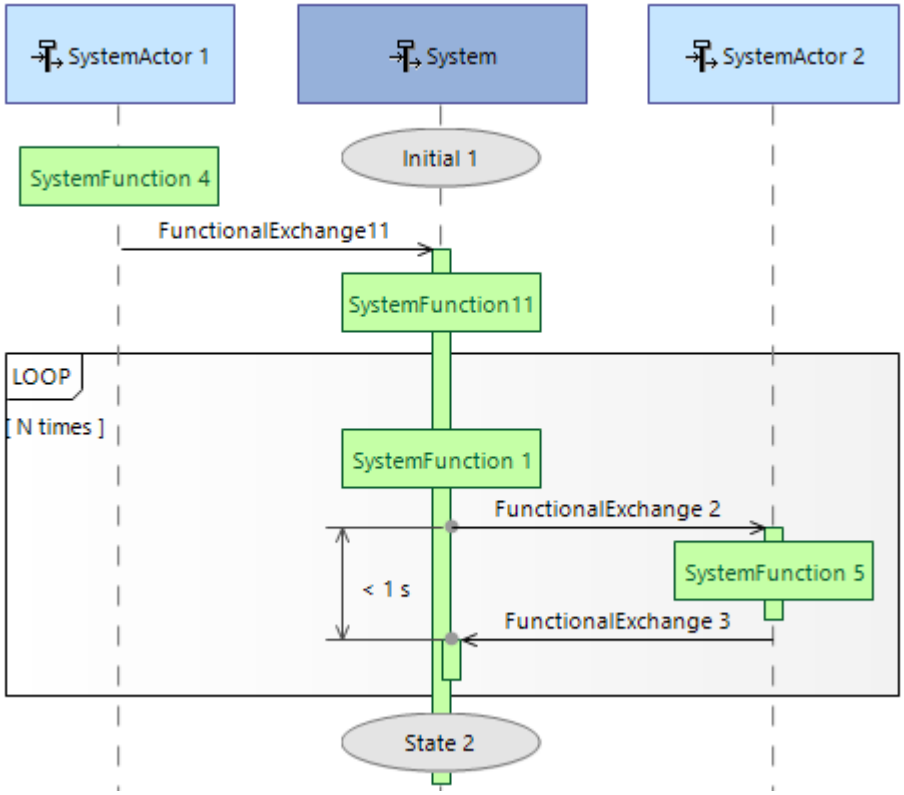
System exchange scenario example with parallel, optional and alternative fragments
The System receives two messages, one from SystemActor 1 (FunctionalExchange 6 from SystemFunction 6) and the other from SystemActor 2 (FunctionalExchange 7 from SystemFunction 9).



- When System receives FunctionalExchange 6, it executes SystemFunction 7 and, if condition a is true, sends Functional Exchange 8 to SystemFunction 8 allocated in SystemActror 1.
- When System receives FunctionalExchange 7, it executes SystemFunction 2 and then, either sends message FunctionalExchange 9 to SystemFunction 10, or sends FunctionalExchange 13 to SystemFunction 13, depending on the value of condition b.

Note: Messages FunctionalExchange 6 and FunctionalExchange 7 may arrive in whatever order, and that the corresponding reactions from System may also happen in whatever order (for instance, SystemFunction 7 may be executed before, after or overlapping the execution of SystemFunction 2). The order of execution is only guaranteed within the two operands (sub-fragments) of the PAR fragment.

System exchange scenario example with loop fragments
SystemFunction 4, allocated to SystemActor 1, sends a message to System, which executes SystemFunction11 and enters a loop.



The loop is executed N times, and inside it, SystemFunction 1 sends a message to SystemActor 2, which executes SystemFunction 5 and responds with another message. It is expected that the interval between FunctionalExchange 2 and FunctionalExchange 3 is less than 1 second. In this particular case, this is a constraint imposed to SystemActor 2. After the loop is executed, the System transitions to State 2.

3.7 Functional chain viewpoint

3.7.1 Description, stakeholders and concerns

Description
The list of definitions related to the concept of a functional chain is listed below:

Functional Chain
Is an ordered set of references to functions and the SPPR-2047 - Functional Exchange that link them, describing one possible path among all the paths forming the data-flow. Functional chains are used to describe the system behaviour in a particular usage context, to contribute to one or more SPPR-2583 - System Capability. Each reference to a function or exchange inside the chain can be qualified by an expectation in the context of the chain (the value that an exchange item or a function attribute should take, for example).

Furthermore: A functional chain also specifies constraints or expectations of precedence or anteriority via oriented sequence links. A set of functions and sequence links is a sequence. Control nodes can be defined between the sequence links, to express the parallelism or alternative between several sequences of functions, or, also the iteration or condition of a sequence to be realised.

ARCADIA talks only about a functional chain from system level onwards, in operational analysis, it is still called operational process, the methodology is the same for both, therefore no further distinction is made.

Viewpoint stakeholders
The list of stakeholders mentioned in SPPR-10950 - Stakeholders adressed is applicable to this viewpoint.

The specific stakeholders/roles addressed by this viewpoint are:

Creator of this viewpoint :

- SPPR-10696 - Systems Engineer as Modeler : Define, analyze and optimize the sequence of functions for the corresponding capabilities

Users of this viewpoint :

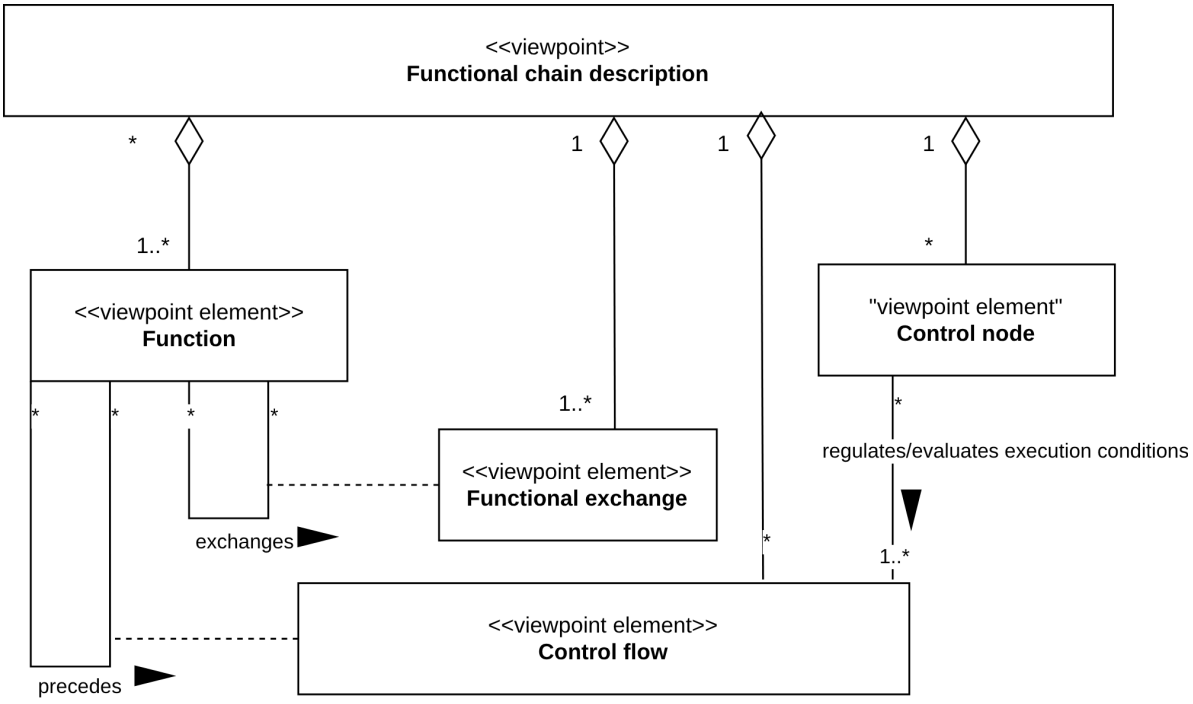
- SPPR-11183 - Test Engineer : Align functional chains with system architecture and validate the system behavior execution logic, ensure the correct implementation of the functional chains in hardware and software.

Concerns
This list of concerns is not exhaustive and can be completed.

- Do functional chains align with the system capabilities and system requirements in terms of traceability, achieving system objectives, etc ?
- Are constraints (e.g. hardware limitations, redundancy, etc) and non-functional requirements (e.g. performance, reliability, maintainability, etc) considered for the functional chains definition?
- Are control and data flows identified for the functional chain?
- How the control and data flows managed accross functions in a functional chains?
- Are dependencies, if existing, between functional chains identified and analyzed?
- How does the system handle errors or failures within a functional chain?

3.7.2 Visualisation kind

Visualisation kind meta model for functional chain



3.7.3 View method

Construction method functional chain
Use Capella Functional Chain Diagrams to model Functional chain viewpoints. See SPPR-10855 - View modelling rules. Capella elements to be used in a functional chain viewpoint are :

All layers

- SPPR-10105 - Exchange Item

Operational Analysis

- SPPR-10222 - Operational Interaction
- SPPR-10220 - Operational Activity

System Need Analysis

- SPPR-10082 - Function (System Function)
- SPPR-10085 - Functional Exchange (System Functional Exchange)

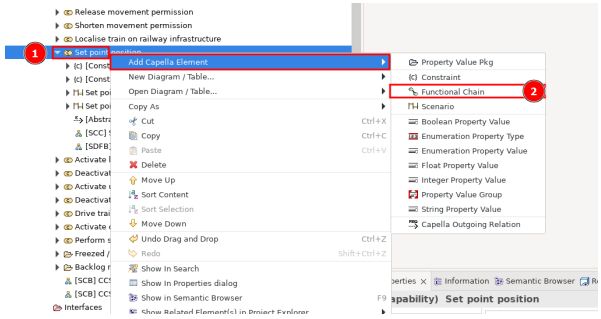
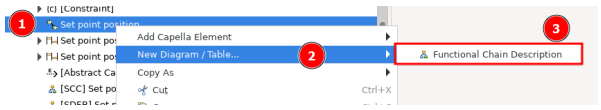
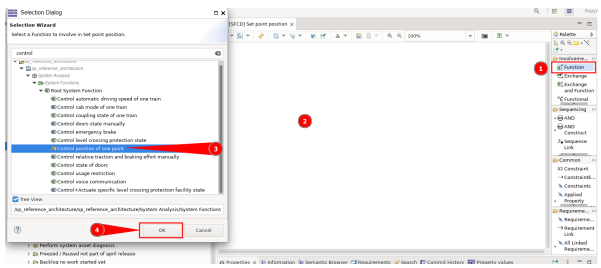
Logical Architecture

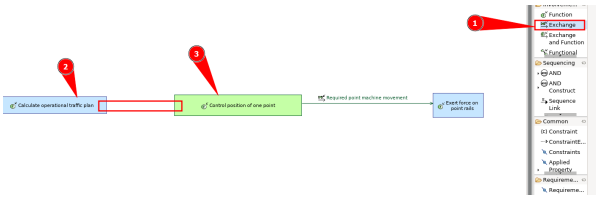
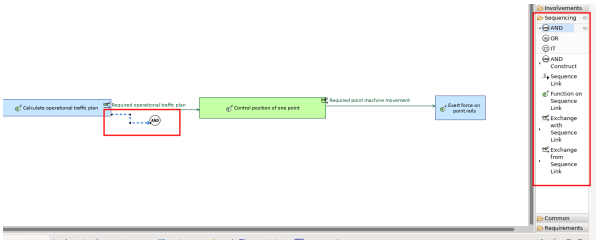
- SPPR-10082 - Function (Logical Function)
- SPPR-10085 - Functional Exchange (Logical Functional Exchange)

Physical Architecture

- SPPR-10082 - Function (Physical Function)
- SPPR-10085 - Functional Exchange (Physical Functional Exchange)

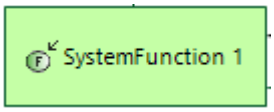
Here is a step by step tutorial to create a functional chain viewpoint in Capella:

Step description	Illustration
1. Create a functional chain for the corresponding capability (in this example we use System Need Analysis modelling level) 1.1 Right-click on the capability in the Project explorer 1.2 Select "Add Capella Element" then "Functional chain" 2. Rename the functional chain according to the naming conventions in SEMP annex M2.	
3. Create a functional chain diagram for the viewpoint 3.1 Right-click on the created functional chain in the Project explorer 3.2 Select "New Diagram / Table" then "Functional chain description" 4. Rename the functional chain diagram according to the naming conventions in SEMP annex M2.	
5. Add functions to the functional chain 5.1 Click on "Function" in the Palette 5.2 Left-click in the white area of the diagram 5.3 Select the function that should be added to the viewpoint 5.4 Left-click ok	

Step description	Illustration
6. Add functional exchanges to the functional chain 6.1 Click on "Exchange" in the Palette 6.2 Left-click on the 1st function to define the source of the Functional Exchange 6.3 Left-click on the 2nd function to define the target of the Functional Exchange 6.4 Select the corresponding functional exchange 6.5 Left-click ok Note : You can only add existing functional exchanges in the functional chain viewpoint. To create a new functional exchange, see SPPR-8867 - Functional flow viewpoint .	
7. If needed, add control node elements to the functional chain 7.1 Click on the control node type in the Palette 7.2 Left-click in the white area of the diagram 734 Insert "Sequence links" from the palette	

Interpretation method functional chain diagrams
Functional chains are expressed by functional chain description diagrams.
Functional chains description diagrams allow the designer to isolate the representation of the functional chain from the global data flow (in *DFB diagram) in order to modify it.

A functional chain contains:
Functional Chain Involvement



The references to functions are called in Capella "functional chain diagrams function involvements". They are not functions themselves but can only be created to represent functions that already exist. A Function that is involved in the Functional Chain. The same Function can be involved several times in the same Functional Chain.

Functional Chain Involvement Link

Usually, Functional Exchange that is involved in the functional chain.

Sequence Link

Sequence Links express a precedence between the execution of the represented Functions. A Sequence Link indicates that in the context of this Functional Chain, the source Function should operate before the target Function. More precisely, the target Function cannot produce its expected output value before receiving the inputs produced by the source Function. A Sequence Link has an optional triggering condition and can be combined with Control Nodes (OR, AND, ITERATE).

Control Nodes (OR, AND, ITERATE)

Control Nodes can be used to enrich a function with additional sequencing information. The control nodes and sequence links can be compared a bit with a functional flow block diagram or SysML v1 activity diagram.

Assembling functional chains
Functional chains can be assembled and reused in order to express more complex scenarios. The definition for the methods is that a functional chain is assembled in another one in case there is an include/extent relationship between the respective owning capabilities. In that case the included or extended functional chain should be connected to the functional chain of interest using sequence links. The included/extended functional chain will be displayed minimised.

3.7.4 View modelling rules

Functional chain description
Describes the chain of functions and exchanges needed to fulfil a capability.

1. one diagram exists for each functional chain
2. only directly involved functions are shown
3. only sequence links necessary to reach the end conditions of the corresponding system capability are shown
4. only directly involved functional exchanges are shown
5. allocated exchange items on functional exchanges are shown instead of the functional exchange names

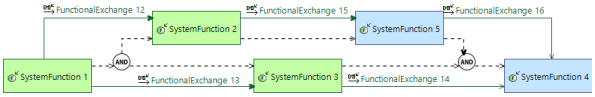
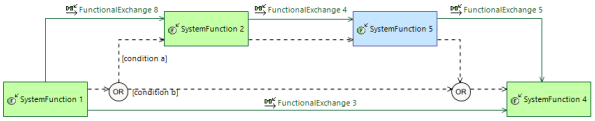
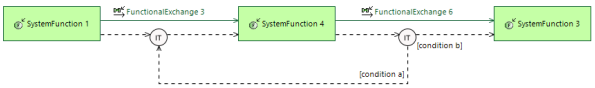
Allow plenty of space so that you can set the diagram out in an orderly way; these diagrams quickly become unreadable if a left-to-right or top-to-bottom convention is not applied.

Notes	[SFCD] [LFCD] [PFCD]
ID	SPPR-8178
Example	• SPPR-7492 - Functional chain examples for capabilities

Control nodes express conditions for a sequence of functions
Control nodes can be of type :

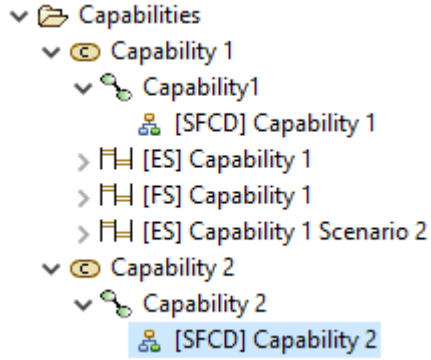
- IT
  - Where the same function or sequence of functions occurs more than once within a functional chain.
  - Where the execution logic of the functional chain requires a return loop from a later function to an earlier function.
- AND
  - Where two or more branches of the process run in parallel/concurrently the AND node is used to split one incoming sequence link into two or more outgoing sequence links.
  - Where two or more branches of the process run in parallel/concurrently the AND node is used to merge two or more incoming sequence links into one outgoing sequence link.
  - This does not mean that AND nodes must be always used as matching pairs where the number of outgoing links from the first AND equals the number of incoming links to the second AND, as long as all parallel branches are eventually terminated by an AND.
- OR
  - Where a decision is made between two or more possible, mutually exclusive paths in the process the OR node is used to split one incoming sequence link into two or more outgoing sequence links.

◦ Where two or more alternative paths come back together to a common path the OR node is used to merge two or more incoming sequence links to one outgoing sequence link.

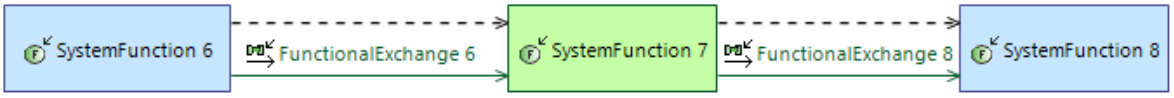
ID	SPPR-11185
Example	Functional chain with AND control node:  • Function 2 AND Function 3 change the output after Function 1 does • Function 2 and 5 as well as function 3 are both providing their outputs in an independent order to functional 4. Functional chain with OR control node:  • Function 2 and 5 are only change the outputs according to the condition attached to their respective sequence link • Function 4 changes the output after Function 1 OR Function 2 + 5 Functional chain with ITERATE control node:  • Function 4 is executed as long as "condition a" is true.

Functional chain examples for capabilities
Functional Chains are very useful to describe the functionality of a Capability. A Capability should contain at least one main Functional Chain, although it could have more than one. A Functional Chain may involve (use) other Functional Chains, even those used to describe the functionality of other Capability.

System with two System Capabilities (Capability 1 and Capability 2), each described by one Functional Chain:

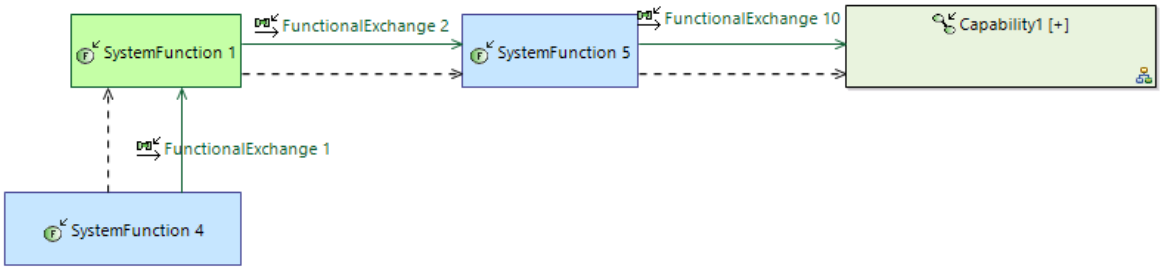


Functional Chain for Capability 1 of the previous example:

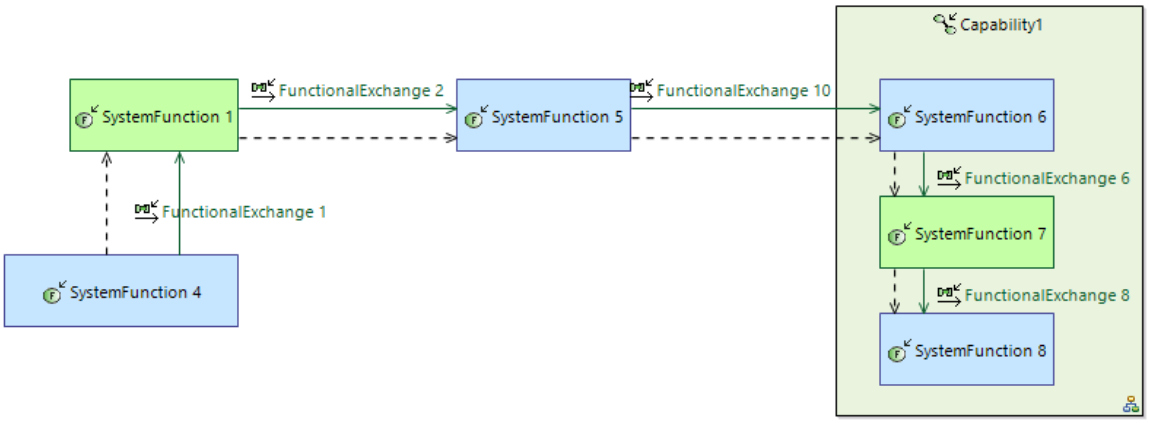


Functional Chain for Capability 2 of the previous example. Note that it involves the Functional Chain

used to describe Capability 1:



The same Functional Chain as above, but expanded to show explicitly the sequence of System Functions in the Functional Chain of Capability 1:



3.8 Control loop viewpoint

3.8.1 Description, stakeholders and concerns

Description
The list of definitions related to the concept of a control loop is listed below. See also SPPR-7485 - Method for definition of control loop functions.

Control loops
The control loops are used in System Pillar activities in order to :

- Define a safety architecture
- Identify the potential interactions with other domains (not exhaustive)
- Ensure that interconnections of control loops are deeply analysed to avoid a cut in a control loop

This viewpoint is essential for systems that depend on regulation, automation, or responsive behaviour such as railway systems.

Generic info
Control Loops are treated as closed functional chains with feedback to the first function. A control loop is a specific sequence of system functions where input (sensed data) is processed to influence an output, and the outcome is monitored to adjust future actions. The feedback from the output is routed back to the initial function, closing the loop.

Viewpoint stakeholders
The list of stakeholders mentioned in SPPR-10950 - Stakeholders adressed is applicable to this viewpoint.

The specific stakeholders/roles addressed by this viewpoint are:

Creator of this viewpoint :

- SPPR-10696 - Systems Engineer as Modeler : Define, analyze and optimize the sequence of functions and functional flow for the corresponding control loop

Users of this viewpoint :

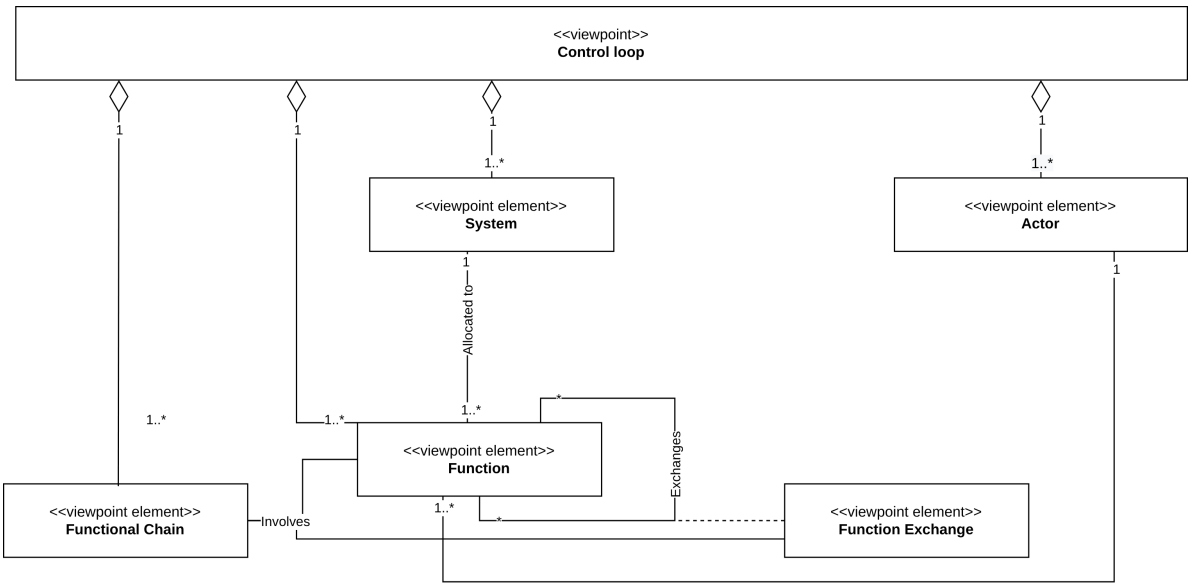
- SPPR-11183 - Test Engineer : Assess correctness, consistency and performance of the control loop. Ensure the correct implementation of the control loop in hardware and software. Ensure alignment between the control loop and system behavior execution logic.
- European Union Agency for Railways (ERA) : Evaluate and validate control loop designs in railway systems to ensure alignment with safety, interoperability, and regulatory requirements defined for the European rail network.

Concerns
This list of concerns is not exhaustive and can be completed.

- Which data flow and functional exchanges are being controlled and monitored?
- How is feedback captured, transmitted and used/processed?
- Are constraints (e.g. hardware limitations, redundancy, etc) and non-functional requirements (e.g. performance, stability, responsiveness, reliability, maintainability, etc) considered for the control loop definition?
- How do control loops interact with other system behaviours or external conditions? Are dependencies, if existing, between control loops identified and analysed?
- Do control loops align with the system capabilities, system requirements and stakeholder needs in terms of traceability, achieving system objectives, satisfying operational needs, etc?

3.8.2 Visualisation kind

Visualisation kind meta model



3.8.3 View method

Construction method control loop
Use Capella Architecture and/or Functional Chain Diagrams to model Control loop viewpoints. See SPPR-10921 - View modelling rules and SPPR-10855 - View modelling rules. Capella elements to be used in a control loop viewpoint are:

System Need Analysis

- SPPR-10060 - Actor (System Actor)

- SPPR-10062 - System (System Component)
- SPPR-10082 - Function (System Function)
- SPPR-10085 - Functional Exchange (System Functional Exchange)
- SPPR-10097 - Functional Chain

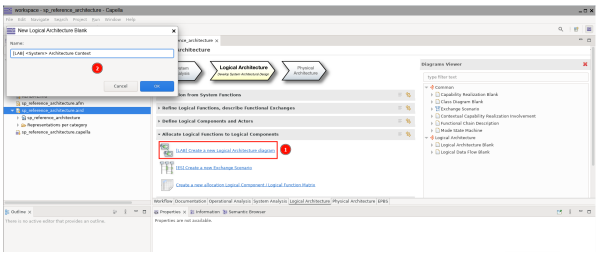
Logical Architecture

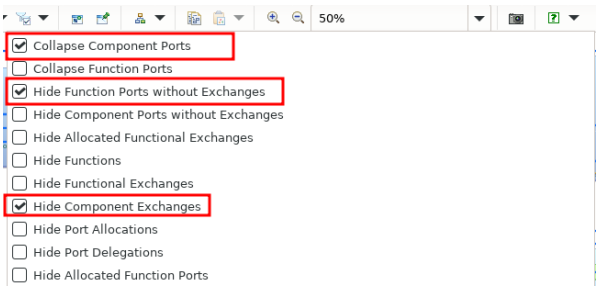
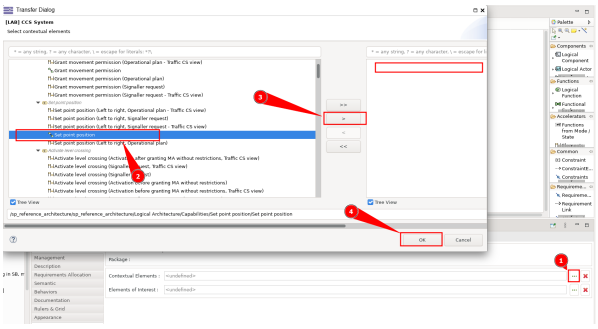
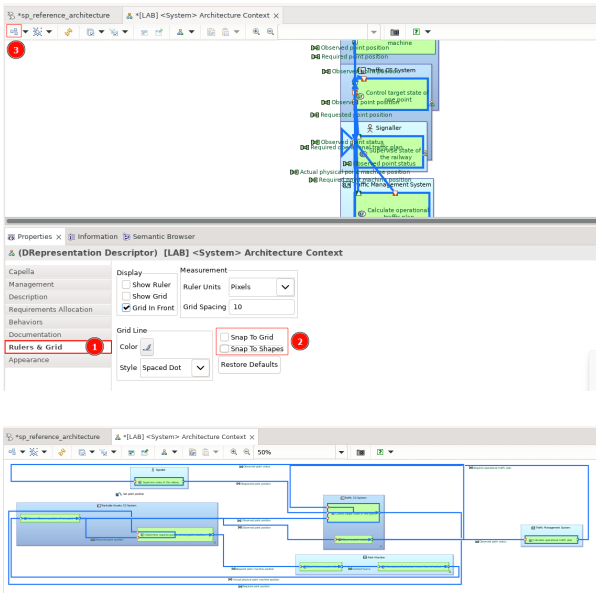
- SPPR-10060 - Actor (Logical Actor)
- SPPR-10062 - System (Logical Component)
- SPPR-10082 - Function (Logical Function)
- SPPR-10085 - Functional Exchange (Logical Functional Exchange)
- SPPR-10097 - Functional Chain

Physical Architecture

- SPPR-10060 - Actor (Physical Actor)
- SPPR-10062 - System (Physical Component (Node or Behaviour))
- SPPR-10082 - Function (Physical Function)
- SPPR-10085 - Functional Exchange (Physical Functional Exchange)
- SPPR-10097 - Functional Chain

Here is a step by step tutorial to create a Control loop viewpoint in Capella based on Functional Chains and Architecture viewpoints:

Step description	Illustration
A functional chain need to be created following the functional chain viewpoint. The functional chain should include all functions for the corresponding control loop.	See SPPR-11021 - Construction method functional chain
1. Create an Architecture Blank diagram for the control loop (in this example we use a LAB for Logical Architecture modelling level) 2. Rename the diagram according to the naming conventions.	

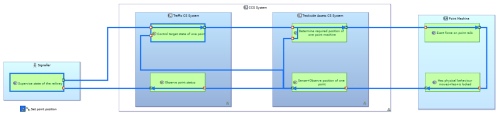
Step description	Illustration
Set filter settings in the diagram inline with modelling rules: • Collapse component ports • Hide functional ports without exchanges • Hide component exchanges • Hide functional exchange names	
Using the diagram properties to add model elements for the corresponding control loop 1. Left-click on contextual elements 2. Select the functional chain or chains for the corresponding control loop (Note first you need to create the functional chain if not yet existing) 3. Move the content to the right 4. Left-click on OK This will add all functions and components for the corresponding control loop to the diagram.	
Improve diagram layout • Select "Rulers & Grid" in the diagram properties • Uncheck "Snap To Grid" and "Snap To Shapes" functionalities • Use diagram auto layout.	

3.8.4 View modelling rules

Control loop

- Architecture diagram is used
- At least one functional chain is represented on the diagram
- The functional chain is set as contextual element

- The following filter are set for the diagram:
  - Collapse component ports
  - Hide functional ports without exchanges
  - Hide component exchanges
  - Hige labes of functional exchanges

Notes	[SAB], [LAB], [PAB]
ID	SPPR-11362
Example	

3.9 Architecture description viewpoint

3.9.1 Description, stakeholders and concerns

Description
The architecture context shows an overview of all structural architecture elements. Refer to other Viewpoints description sections for definitions.

Viewpoint stakeholders
The list of stakeholders mentioned in SPPR-10950 - Stakeholders adressed is applicable to this viewpoint.

The specific stakeholders/roles addressed by this viewpoint are:

Creator of this viewpoint :

- SPPR-10696 - Systems Engineer as Modeler : Define the system structure for <structural element name> and its interactions, align the viewpoint with the upper system or operational level architecture and the upper level requirements

Users of this viewpoint :

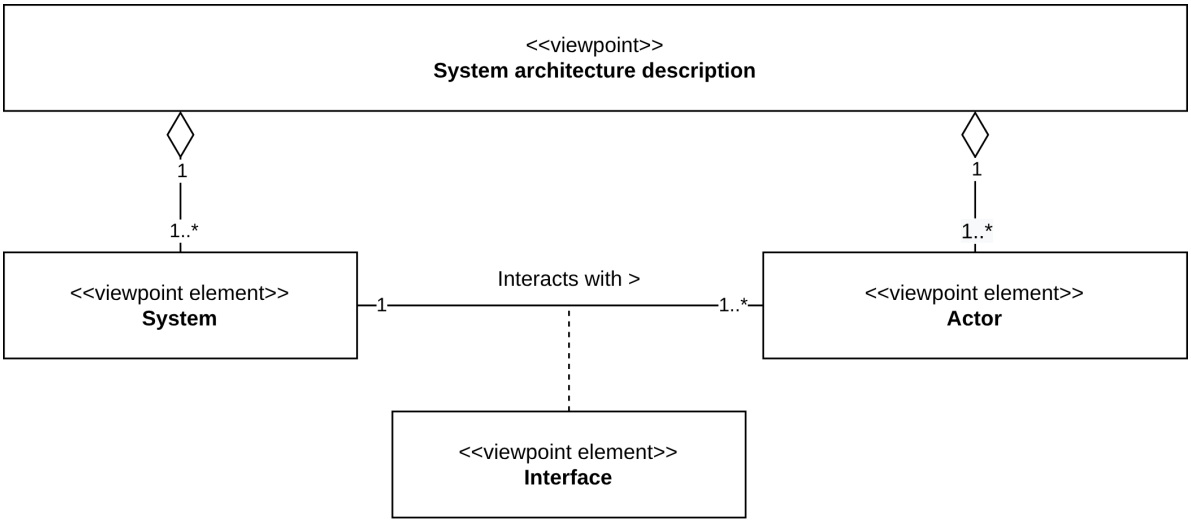
- SPORG-87 - Environment Engineering Team : Ensure consistency with the modelling methods and architecture trade-off principles defined within the SP SEMP
- SPORG-93 - T2 ARC : Ensure functional and technical consistency with the overall System Pillar architecture, evaluate the architectural decisions and validate the viewpoint
- SPPR-11183 - Test Engineer : Use the system architecture overview for integration purposes and to verify the correctness and completeness of the system behaviour with its environment.

Concerns
This list of concerns is not exhaustive and can be completed.

- Are the system components and their interactions, relationships and dependencies clearly defined ?
- Does the architecture overview satisfy the tradeoff and granularity principles ?
- Does the architecture overview satisfy the CBO ?
- Are functions correctly distributed accross the system, system components or actors ?
- Are constraints (e.g. hardware limitations, SW, etc) and non-functional requirements (e.g. performance, reliability, maintainability, etc) considered for the architecture overview definition ?
- Is the level of detail of the architecture appropriate for the analysis level of the system ?
- Is the arcitecture overview aligned with the system concept and upper level requirements ?
- How does the architecture affect scalability and future system upgrades ?

3.9.2 Visualisation kind

Visualisation kind meta model system architecture



3.9.3 View method

Construction method system architecture description
Use Capella Architecture diagrams to model the architecture overview. See SPPR-10921 - View modelling rules. Capella elements to be used in an architecture context viewpoint are:

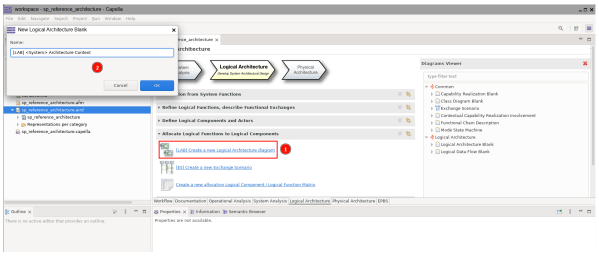
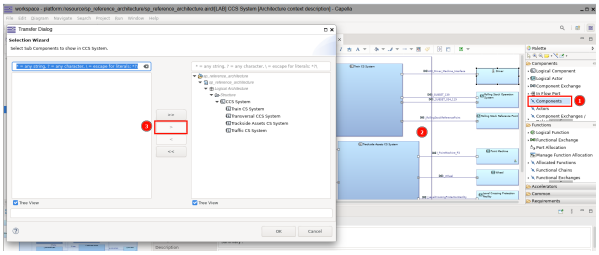
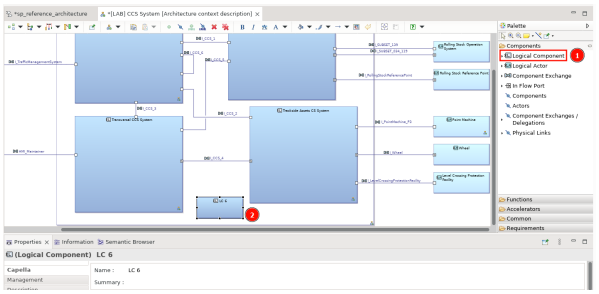
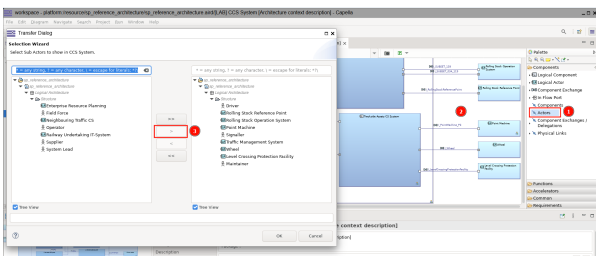
Logical Architecture

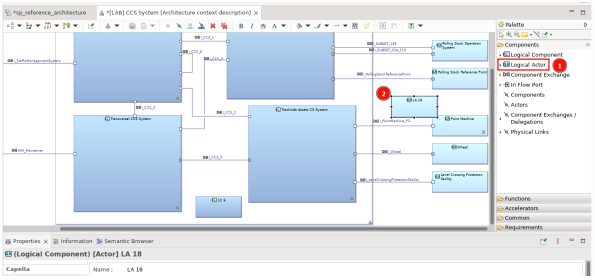
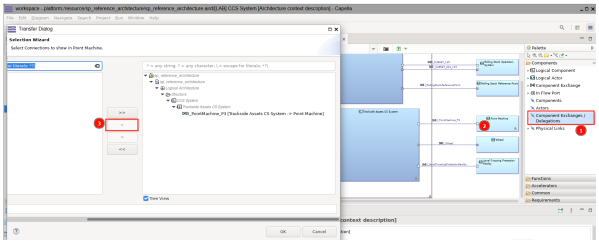
- SPPR-10060 - Actor (Logical Actor)
- SPPR-10062 - System (Logical Component)
- SPPR-10071 - Interface (Logical Component Exchange)

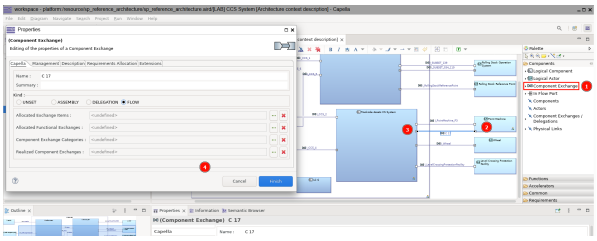
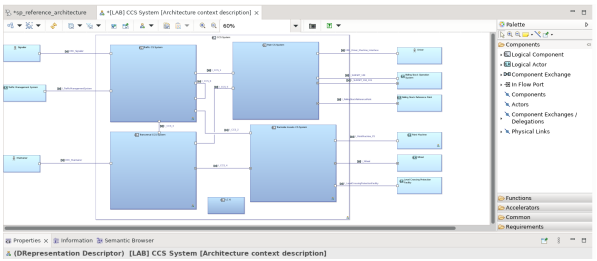
Physical Architecture

- SPPR-10060 - Actor (Physical Actor)
- SPPR-10062 - System (Physical Component (Node or Behaviour)
- SPPR-10071 - Interface (Physical Component Exchange)

Here is a step by step tutorial to create an Architecture description viewpoint in Capella:

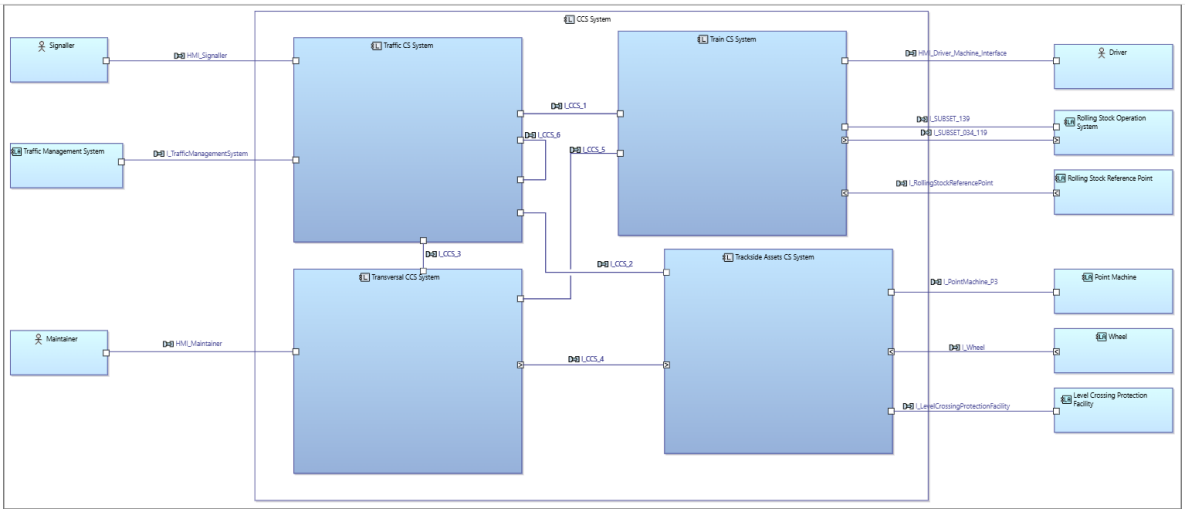
Step description	Illustration
1. Create an Architecture Blank diagram for the Architecture description (in this example we use a LAB for Logical Architecture modelling level) 2. Rename the diagram according to the naming conventions in SEMP annex M2.	
3. Using the Palette, Insert all the existing (or create new) logical components within the logical system block **Existing components** To add the existing logical components : - Left-click on "Components" - Left-click in the the logical system block (If not existing, then create it the same way you create logical components. See below) - Select the components that should be added to the viewpoint - Move it to the right field - Left-click ok **New components** To add new logical components : - Left-click on "Logical component" - Left-click in the logical system block area (If not existing, then click in the white area of the diagram)	 
4. Using the Palette, Insert all the existing (or create new) Actors in interaction with the logical components **Existing actors** To add the existing logical actors : - Left-click on "Actors" - Left-click in the white area of the diagram - Select the actors that should be added to the viewpoint - Move it to the right field - Left-click ok **New actors** To add new logical actors :	

Step description	Illustration
- Left-click on "Logical Actor" - Left-click in the white area in the diagram	
If the Interface (**Component Exchange**) has already been created in another diagram (e.g. System Interface description): 5. Using "Component exchanges" in the Palette and by selecting the Logical component or Actor, display the component exchange between the entities. To add the existing component exchange : - Left-click on "Component exchanges / Delegations" - Left-click on the logical component or actor for whom you would like to add the interface - Move the component exchange to the right field - Left-click ok Go directly to <u>Step 7</u> below.	

Step description	Illustration
If not : 5. Using "Component exchange" in the Palette and by selecting the Logical component then Actor (or vice versa), create a component exchange between the entities. 6. Rename the component exchange and allocate functional exchanges accroding to the modelling rules for SPPR-10852 - Interface viewpoint)	
7. Rearrange the diagram to make it readable	

Interpretation method
An Architecture description viewpoint is particularly useful to define the constitutive elements of the system on logical and physical level. It also allows to represent an overview of the **internal interfaces** between components within a system and **external interfaces** between the system and external actors.
It is also useful to show the allocation of functions to components or actors (see SPPR-10851 - Function allocation viewpoint).

Example of an Architecture description diagram (Logical Architecture level):



In this example, we can first of all see the **structural elements** (blue rectangles), i.e. the Logical Components (contained in overarching box that represents the System at the Logical level) and the Actors.

The interfaces between the system components and the actors are represented by **Component Exchanges** (blue line). A Component Exchange either links the Logical System to one of its Actors, or a Logical Component directly to an external Actor, or two Logical Components via Component Ports (uni- or bidirectional; white squares).

3.9.4 View modelling rules

Architecture description
Describes the architecture of the system including its system elements (ex. logical components) as well as external and internal interfaces.

- one diagram exists for the architecture level of the system of interest
- all systems shall be shown
- all actors shall be shown
- all interfaces (component exchanges) shall be shown

Notes	[LAB] [PAB]
ID	SPPR-8378

3.10 State machine viewpoint

3.10.1 Description, stakeholders and concerns

Description
The list of definitions related to the concept of state machine is listed below:

State Machine
Collection of states connected by transitions.

State
An observable and measurable attribute used to characterise a behaviour of an entity during which some invariant condition holds. States can be applied to elements such as actors, systems or subsystems.

State Transition
A change of state toward itself or toward another state. A transition is characterised by a trigger which can be an event or a boolean condition.

Viewpoint stakeholders
The list of stakeholders mentioned in SPPR-10950 - Stakeholders adressed is applicable to this viewpoint. The specific stakeholders/roles addressed by this viewpoint are:

Creator of this viewpoint :

- SPPR-10696 - Systems Engineer as Modeler : Define and analyze system states, modes and transitions

Users of this viewpoint :

- SPORG-81 - T2 Transversal - SPORG-1711 - T5 HERD : Consider the system states in the definition of the Data model for Diagnosis & Maintenance purpose
- SPPR-11183 - Test Engineer : Ensure the correct implementation of the system modes/states in hardware and software, validate the system behavior in various states.

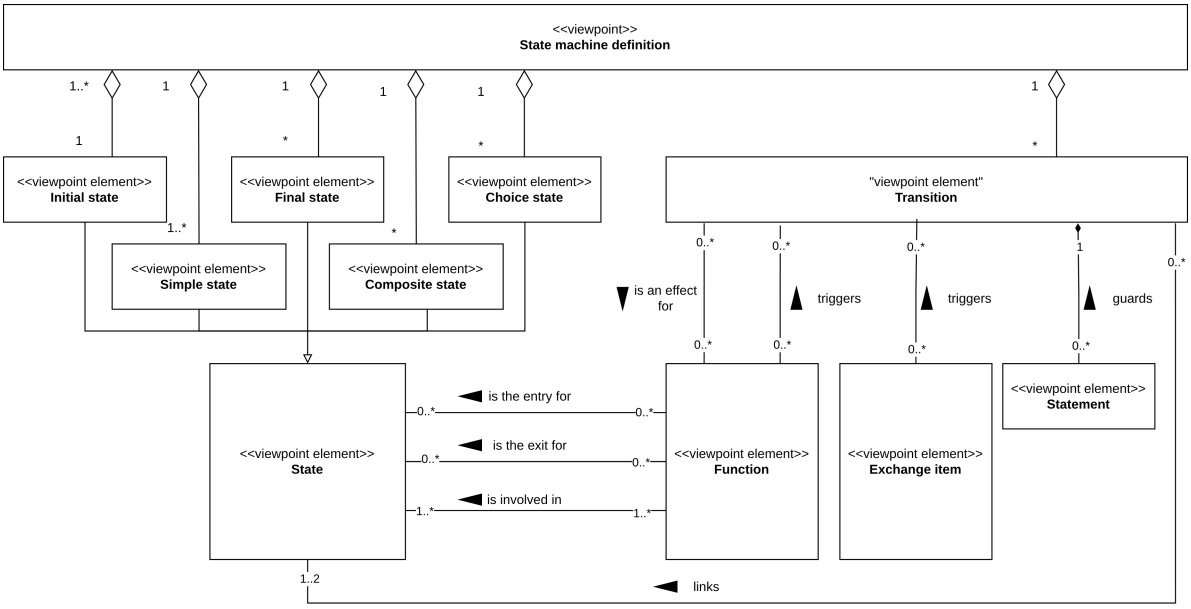
Concerns
This list of concerns is not exhaustive and can be completed.

- Are all possible system states/modes identified?
- Are all conditions, triggers and events correctly defined for the state transitions?

- How does the system handle abnormal or failure states?
- Are constraints (e.g. timing, redundancy, etc) and non-functional requirements (e.g. performance, reliability, maintainability, etc) considered for the state machine definition?
- How can state-based behaviour be tested and validated at a later stage?

3.10.2 Visualisation kind

Visualisation kind meta model for state machine



3.10.3 View method

View method
Will be provided in a future version of this document.

3.10.4 View modelling rules

View modelling rules
Will be provided in a future version of this document.

3.11 Information model viewpoint

3.11.1 Description, stakeholders and concerns

Description
The list of definitions related to the concept of state machine is listed below:

Information model
Is a high-level representation of the information needed to describe and/or operate a system. It focuses on concepts, their relationships, and their properties.

Data Class
A data class is a structure that defines the data and their elementary properties.

Viewpoint stakeholders
The list of stakeholders mentioned in SPPR-10950 - Stakeholders adressed is applicable to this viewpoint. The specific stakeholders/roles addressed by this viewpoint are:

Creator of this viewpoint :

- SPPR-10696 - Systems Engineer as Modeler : Define and maintain the system's data model

Users of this viewpoint :

- SPORG-81 - T2 Transversal - SPORG-1711 - T5 HERD : Consider the system data model in the definition of the overall System Pillar Reference Architecture information model (CCS-TMS data model)
- SPORG-93 - T2 ARC : Ensure functional and technical consistency with the overall System Pillar information model and ERA information model, validate the viewpoint
- SPPR-11183 - Test Engineer : Ensure the correct implementation of the system data model in hardware and software, validate correctness and completeness of the system data model

Concerns
This list of concerns is not exhaustive and can be completed.

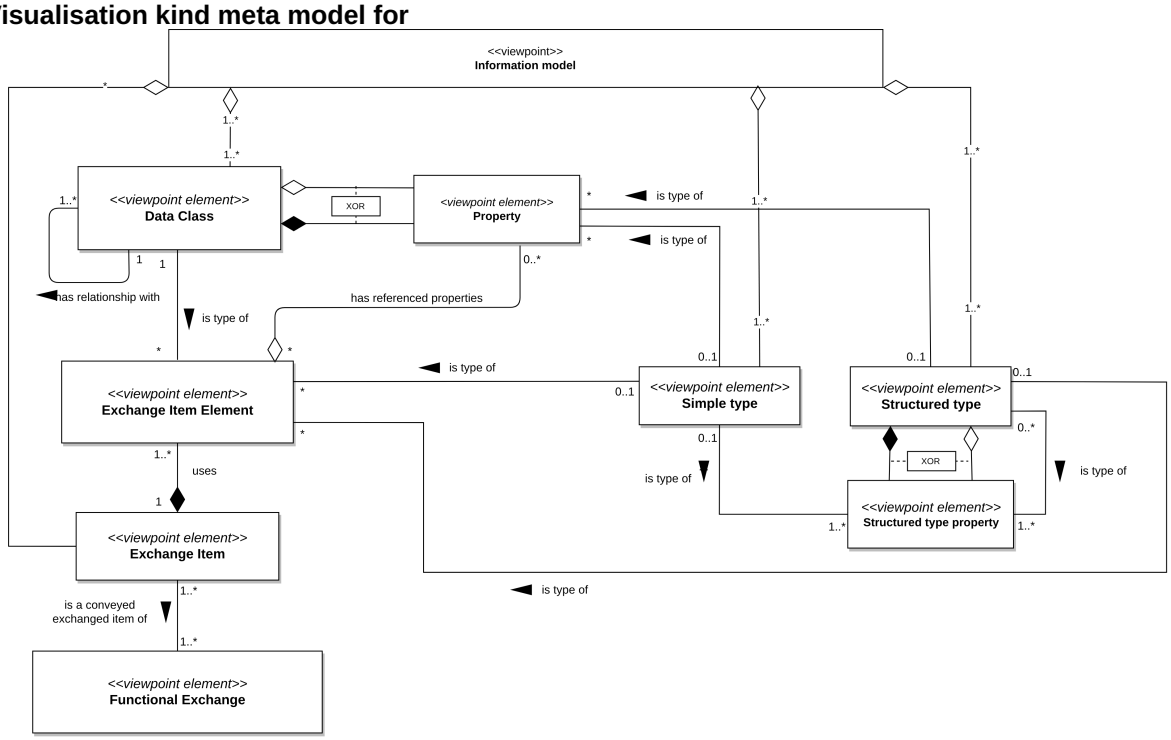
- Are all system data elements and their relationships identified and correctly structured ?
- Are the information flow patterns identified across system elements ?
- How is data consistency, integrity and security ensured ?
- Are constraints (e.g. timing, redundancy, etc) and non-functional requirements (e.g. performance, reliability, maintainability, etc) considered for the information model definition?
- How can the information model be extended for future requirements ?

Added values of Information model viewpoints :

- The possibility to anticipate resource problems before they occur;
- An additional guarantee of the quality, security, and accessibility of data;
- Better allocation of resources;
- Reinforcing the communication, relationship, and collaboration between the functions, logical components and subsystems.

3.11.2 Visualisation kind

Visualisation kind meta model for



3.11.3 View method

Construction and interpretation method information model capella

General Approach

The conceptual data model expresses an overview of railway concepts without going into detail. It is therefore completely independent of technologies, systems, and software.

It is used as a starting point for discussion between the project stakeholders. The important categories of data classification are selected and the relationships between them are described. These classes should be based on concrete cases. Everyone should be able to understand the established conceptual model. Technical details are reserved for the following two types of models.

Modification of data object classes

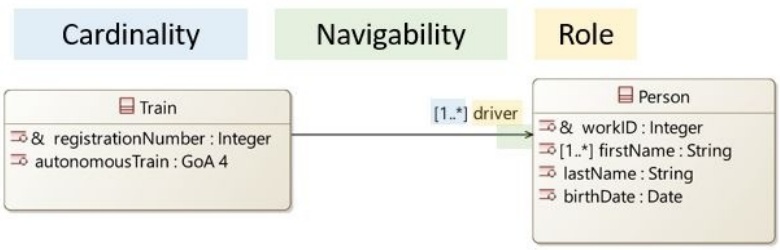
Data object classes must be aligned with any international collaborations. Therefore, this task may be done jointly with other railway organisations if it makes sense to do the work once jointly and then adopt the outcome for ourselves.

Relationships between classes

General information

Before identifying the different possibilities of relationships between classes, it is worth recalling some general characteristics:

- Role: define the relationship end (most of the time it is the name of the pointed class)
- Navigability: means that the instance of the pointed class is accessible from the class of the pointing instance. Both sides of a link may be navigable, but most of the time there is only one side or no side at all. An open arrowhead on the end of an association indicates if the end is navigable.
- Link properties : relationships between classes have properties related to cardinality (ex. multiplicity).

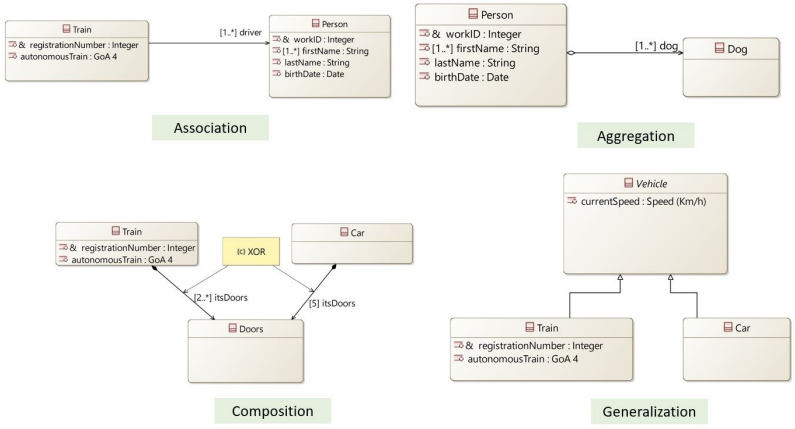


Capella info: Be careful, navigability (arrows on associations) and direction of the verb naming the association are completely distinct notions.

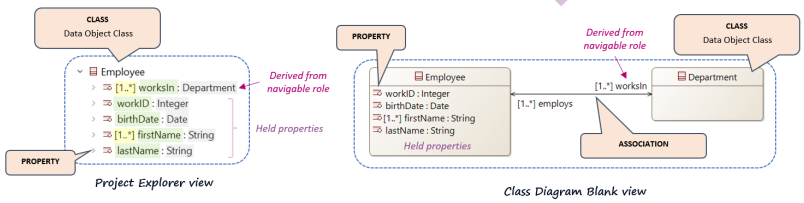
Specific use cases

Relationship	Cases
AGGREGATION	A variant of the "has a" association relationship (represented by a white diamond)
ASSOCIATION	Represents a relationship shared among the instances of two classes. (represented by a link between classes)

Relationship	Cases
COMPOSITION	A form of aggregation that requires a part instance to be included in a composite. If a composition is deleted, all its parts are normally deleted with it. Be careful if there are multiple compositions in which the same class is involved as a part. (represented by a black diamond)
GENERALIZATION	Also called realisation relationship: where a generalization relates a specific class to a general class, each instance of the specific class is also an instance of the general class. Therefore, properties specified for instances of the general class are implicitly specified for instances of the specific class. (represented by a hollow triangle).



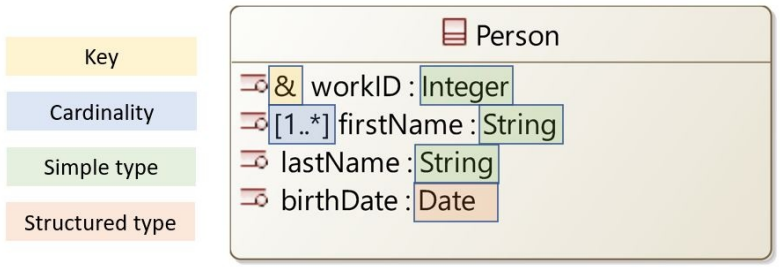
General information on properties
A property can be held directly by a class (this is called an attribute) or can be derived from a navigable role through an association link (normally already defined in the conceptual data model).



The principal characteristics of properties are:

- Type: come from other classes or simple/structured types.
- Key: It means that the values of this property enable the distinction of different instances of the class. It is noted with a "&" prefix.
- Cardinality

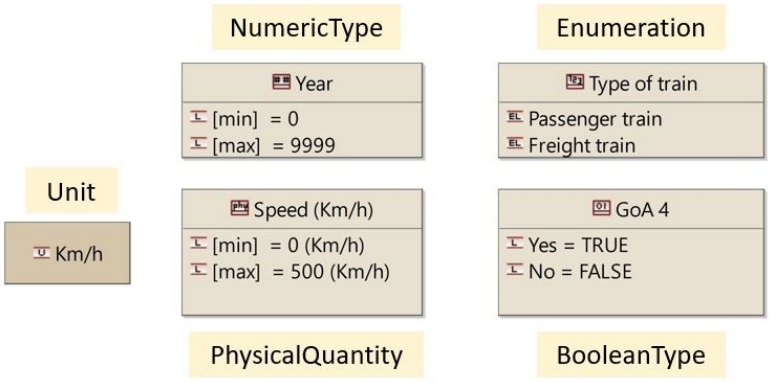
In order to define the type of the data object class, it is necessary to create other classifiers: simple type and structure type. /!\ It is not possible to use a data object class to type the property of another data object class.



Simple data type

The purpose is to identify types with simple data (without properties). There exists different kinds of simple types:

- NumericType: represents numbers (float or integer)
- PhysicalQuantity: represents a physical dimension and is associated with a unit (meter, gram, volt, etc.)
- Enumeration: represents possible values which are enumerated (can be defined with EnumerationLiteral)
- BooleanType: represents named values which hold the semantics of truth or falsehood (can be defined with BooleanLiteral)
- StringType: represents a sequence of characters

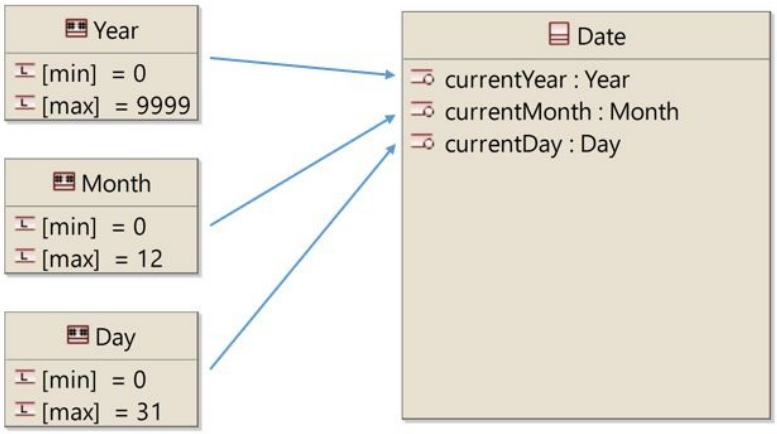


Warning: Simple types cannot have properties. If there is a need for structured data types we have to create a structured type

Structured data type

The purpose is to identify types with complex data because they can have properties (simple types or other structured types) and can be used as a type of property in another structured type. The difference between a structured type and a data object class is that a structured type has no roles (i.e there are no association links between structured types).

Note: instances of a data type (simple or structured) are identified only by their value, which means that data types with the same value are considered to be equal instances.



Capella info:
In Capella, they are considered primitive classes, which means that these classes will have no identity in the context of the system and cannot be the source or target of association links. Two instances of this class with the same property values cannot be distinguished (e.g. a date does not have any intrinsic identity).
If the primitive aspect of the class is not checked, the structured type cannot be used to type a property of a data object class

Difference between Structured Type and Data Object Class
What is the difference between these two apparently similar objects? Why not simplify the method and use only one of the two concepts? The following table summaries the differences between the concepts:

Question	Structured Type	Data Object Class
Can be used to represent a list of values (enumeration list, boolean values, etc.).	yes	No
Can be used to type a data object class property	Yes	No
Can be used to type an exchange item element	Yes	Yes
Can have relationships and roles with other elements	No	Yes

In other words, both concepts are essential to the development of the data model. Data object classes represent the concepts and objects that will be transmitted, used, consumed and read throughout the system. But in order to define the types of their properties or the different possibilities of values, structured types appear to be indispensable.

3.11.4 View modelling rules

View modelling rules
Will be provided in a future version of this document.

4 Diagram naming convention

Define diagram name
Each diagram shall be named according to the following convention:

- **[<diagram type abbreviation>] <subject information> [<view name>]**

part	explanation
diagram type abbreviation	The defined abbreviations of the diagrams are defined here: SPPR-8386 - Mapping of the System Need Analysis diagrams to specific diagram-based views SPPR-8385 - Mapping of the logical architecture diagrams to specific diagram-based views SPPR-8401 - Mapping of the physical architecture diagrams to specific diagram-based views
subject information	consists of either the name of a considered **<model element>** or <**free text>**
	<model element> is the exact name of the considered model element • used if the view focuses on a single model element
	<free text> is a useful title providing initial information about the diagram content • used if the view does not focus on a single model element or for other cases (e.g. set of model elements shown) • this shall be written in sentence case - the first letter of the first word in a sentence is capitalised
view name	name given to the view according to the view modelling rules in this guideline

ID	SPPR-6846

Explanation of diagram-based views
A diagram-based view is a specific diagram which is mapped to a Capella default diagrams. In other words, the default diagrams are tailored to a specific purpose including the diagram rules to enforce those purposes.

The existing default diagram types are also specifically tailored regarding their name and abbreviation. Example: An exchange scenario on the logical architecture is called "Logical Exchange Scenario" with the abbreviation "LES". By default, Capella just calls them "Exchange Scenario" and "ES" independent from the current considered ARCADIA layer. The following diagram types and names are modified in context of this guideline:

- Class Diagram Blank (CDB) - all layers
- Mission and Capabilities Blank (MCB) - System Needs Analysis only
- Contextual Capability (CC) - System Needs Analysis only
- Exchange Scenario (ES) - all layers
- Modes and States (MSM) - all layers
- Capability Realization Blank (CRB) - Logical Architecture and Physical Architecture only

4.1.1 Views for the System Need Analysis

Mapping of the System Need Analysis diagrams to specific diagram-based views
The following mapping provides an overview of the used default diagrams, their tailored abbreviations and the corresponding specific views. The diagram abbreviation is part of the name of each specific view to provide a practical indication which diagram shall be used.

Abbreviation	Default diagram	Specific views	Example
SCB	System Capability Blank	SPPR-6629 - Capabilities definition	[SCB] CCS System [Capability definition]
SCC	System Contextual Capability	SPPR-6630 - System capability context	[SSC] Set point position [Capability context]
SDFB	System Data Flow Blank	SPPR-9580 - Functional flow consolidation SPPR-6633 - Functions involvement	[SDFB] Localise train on railway infrastructure [Functions involvement]
SFBD	System Function Breakdown	currently not used	
SFCD	System Functional Chain Description	SPPR-8178 - Functional chain description	[SFCD] Shorten movement permission [Functional chain description]
SES	System Exchange Scenario	SPPR-8176 - Exchange scenario	[SES] Grant movement permission (Signaller request) [Exchange scenario]
SAB	System Architecture Blank	SPPR-6427 - System context description SPPR-6626 - Interface description SPPR-6628 - Function allocation SPPR-11362 - Control loop	[SAB] CCS System [System context description] [SAB] Activate and Deactivate usage restriction [Control loop]
SSM	System State Machine	currently not used	
SCDB	System Class Diagram Blank	SPPR-8177 - Exchange items description	[SCDB] Perform train movement [Exchange items description]

4.1.2 Views for the Logical Architecture

Mapping of the logical architecture diagrams to specific diagram-based views
The following mapping provides an overview of the used default diagrams, their tailored abbreviations and the corresponding specific views. The diagram abbreviation is part of the name of each specific view to provide a practical indication which diagram shall be used.

Abbreviation	Diagram type	Specific views	Example
LCRB	Logical Capability Realization Blank	currently not used	
LAB	Logical Architecture Blank	SPPR-8378 - Architecture description SPPR-11362 - Control loop SPPR-6626 - Interface description SPPR-6628 - Function allocation	[LAB] Activate and Deactivate usage restriction [Control loop] [LAB] CCS System [Function allocation]

Abbreviation	Diagram type	Specific views	Example
LDFB	Logical Data Flow Blank	SPPR-6633 - Functions involvement SPPR-9580 - Functional flow consolidation	[LDFB] Activate level crossing [Functions involvement]
LFBD	Logical Function Breakdown	currently not used	
LFCD	Logical Functional Chain Description	SPPR-8178 - Functional chain description	[LFCD] Deactivate level crossing [Functional chain description]
LCBD	Logical Component Breakdown	currently not used	
LES	Logical Exchange Scenario	SPPR-8176 - Exchange scenario	[LES] Perform train movement (Full supervision) [Exchange scenario]
LSM	Logical State Machine	currently not used	
LCDB	Logical Class Diagram Blank	SPPR-8177 - Exchange items description	[LCDB] Provide track vacancy detection [Exchange items description]

4.1.3 Views for the Physical Architecture

Mapping of the physical architecture diagrams to specific diagram-based views
The following mapping provides an overview of the used default diagrams, their tailored abbreviations and the corresponding specific views. The diagram abbreviation is part of the name of each specific view to provide a practical indication which diagram shall be used.

Abbreviation	Diagram type	Specific views	Example
PCRB	Physical Capability Realization Blank	currently not used	
PAB	Physical Architecture Blank	SPPR-8378 - Architecture description SPPR-6427 - System context description SPPR-11362 - Control loop SPPR-6626 - Interface description	[PAB] Trackside Assets Control Supervision [Functions allocation] [PAB] Set point position (Operational plan) [Control loop] [PAB] CCS System [System architecture overall description]
PDFB	Physical Data Flow Blank	SPPR-6633 - Functions involvement SPPR-9580 - Functional flow consolidation	[PDFB] Perform train movement [Functions involvement]
PFBD	Physical Functional Breakdown	currently not used	
PFCD	Physical Functional Chain Description	SPPR-8178 - Functional chain description	[PFCD] Perform train movement [Functional chain description]

Abbreviation	Diagram type	Specific views	Example
PCBD	Physical Component Breakdown	currently not used	
PES	Physical Exchange Scenario	SPPR-8176 - Exchange scenario	[PES] Grant movement permission [Exchange scenario]
PSM	Physical State Machine	currently not used	
PCDB	Physical Class Diagram Blank	SPPR-8177 - Exchange items description	[PCDB] Set point position [Exchange items]

5 Appendix

5.1 Standards and References

Standards and References - SEMP Annex M2

Name	Body	Description
SPPR-5911 - [ARCADIA]	AFNOR XP Z67-140	Information technology - ARCADIA - Method for systems engineering supported by its conceptual modelling language - General description - Specification of the engineering definition method and the modelling language
SPPR-11473 - Model-based System and Architecture Engineering with the Arcadia Method	ISTE	This book describes the fundamentals of the method and its contribution to engineering issues such as requirements management, product line, system supervision, and integration, verification and validation (IVV). It provides a reference for the modeling language defined by Arcadia.
SPPR-11474 - Systems Architecture Modeling with the Arcadia Method	ISTE	*Systems Architecture Modeling with the Arcadia Method* is an illustrative guide for the understanding and implementation of model-based systems and architecture engineering with the Arcadia method, using Capella, a new open-source solution.

5.2 Viewpoint Template

Viewpoint Template

Viewpoint Name	**Viewpoint name** A clear and concise name for the viewpoint which indicates the specific perspective needed
Description	A brief and abstract description of the viewpoint. • what is provided by this viewpoint • what is the goal of this viewpoint
Stakeholders addressed	Provide the list of stakeholders (example : architects, designers, decision-makers, etc) who need this viewpoint Describe the purpose/goal of each stakeholder behind this viewpoint Creator of the view based on this viewpoint: Identify who is responsible for constructing the view.

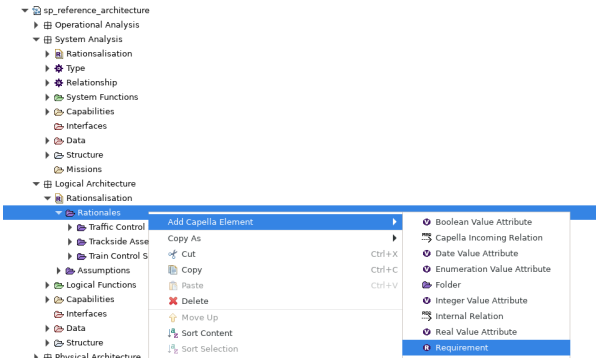
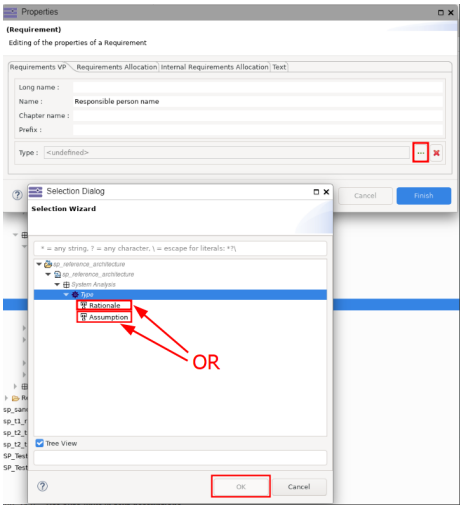
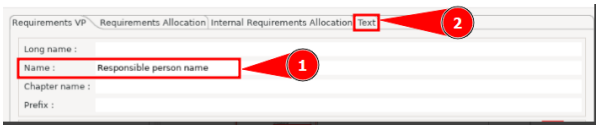
Viewpoint Name	**Viewpoint name** A clear and concise name for the viewpoint which indicates the specific perspective needed
	Reader of the views based on this viewpoint: Specify the intended audience of the view.
Concerns	What key architectural concerns does this viewpoint address? Provide a listing of architecture-relevant concerns to be framed by this architecture viewpoint
Visualisation kind	*1. Visualisation kind languages or notations* *Identify or define the notation used in visualisations of the kind.* *Identify an existing notation or model language or define one that can be used for models of this visualisation kind. Describe its syntax, semantics, tool support, as needed.* *2. Visualisation kind metamodel* *A metamodel presents one or more constructs which are the AD elements that comprise the vocabulary of the model kind and its specification. There are various ways of representing metamodels. The metamodel will present:* • ***entities*** *What are the major sorts of conceptual elements that are present in visualisations of this kind?* • ***attributes*** *What properties do entities possess in visualisations of this kind?* • ***relationships*** *What relations are defined among entities in visualisations of this kind?* *3. Visualisation kind templates* *Provide a template or form specifying the format or expected content of view components governed by this visualisation kind specification.* *Each such template, form, or their parts, can have a legend to be used when this visualisation kind is used within an architecture description.* ——— Define the conventions used for this viewpoint • Classes (which notations, icons to be used in the Capella’s palette) • Attributes (list of properties) • Associations • Extension relationships between metaclasses : for which capella element (Annex M1) a class should be associated/applied, for which capella diagram (Annex M2) a viewpoint should be available (extended diagram)
View methods	View methods are divided into categories, including: • ***construction methods*** *are the means by which views are prepared using a viewpoint. These can be in the form of process guidance (how to start, what to do next); or description guidance (templates for views of this type); or heuristics, styles, patterns, or other idioms to employ.* • ***interpretation methods*** *are the means by which views are to be understood by stakeholders and other users.*
Modelling rules	Document any modelling rules associated with the visualisation kind.
Example	Provide examples to help/guide the user

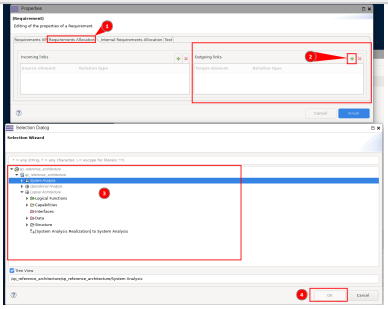
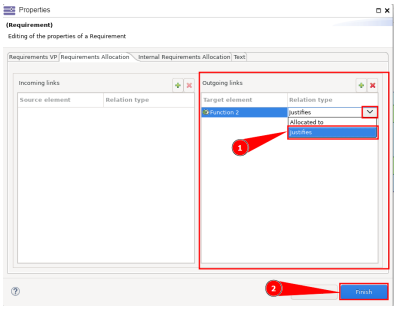
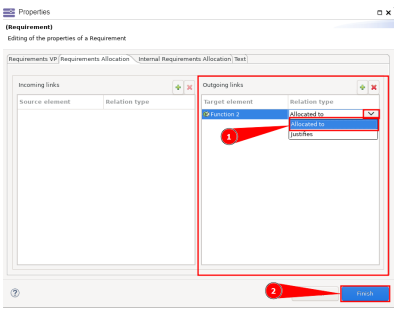
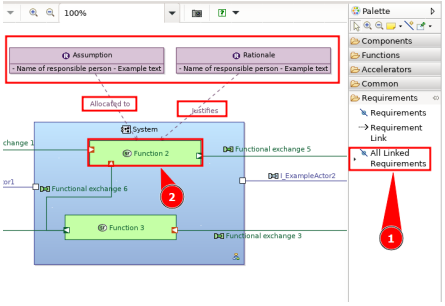
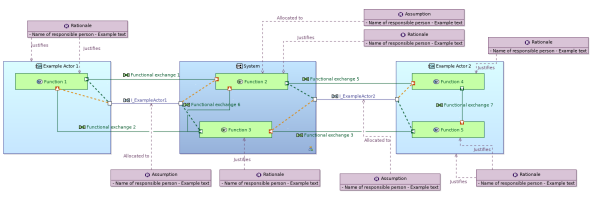
5.3 Cross cutting capella how to's

5.3.1 Rationales and assumptions

How to create rationale and assumptions in Capella

Steps:

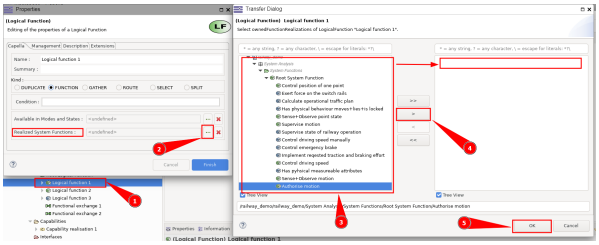
Description	Illustration
Have a look on the modelling rules for ❗ SPPR-3438 - Rationales are justifying design decisions for model elements and ❗ SPPR-10052 - Assumptions are allocated to model elements.	-
To create a rationale or assumption: 1. Right-click on the rationale or assumption package in the relevant architecture level, add a new element by selecting the element requirement (we use the element for rationale and assumption).	
Select the type of the created element: • Either Rationale or Assumption.	
Include the details: 1. Include the name of the responsible person in the field "Name". 2. Include the description into the "Text" tab - If a description is documented outside of the model (e.g. in a Polarion document) put a hyperlink to the current version.	
Select the link of the rationale with an outgoing link of type "Justifies" to all architecture elements which	

Description	Illustration
it pertains. 1. Left-click requirement allocation in the properties of the rationale or assumption. 2. Left-click on the outgoing. 3. Search all relevant model elements that you want to link with the rationale or the assumption.	
Select Rationale: 1. Select the outgoing link of type "Justifies". 2. Finish.	
Select Assumption: 1. Select the outgoing link of type "allocate to". 2. Finish. Note: At the moment there is a bug in Capella where sometimes the link does not show up after clicking finish. What could help: 1. Left-click on the + again after selection, then cancle the elements overview and then finish and safe.	
If useful rationale and assumptions can be included into diagrams: 1. Left-click into the Palette to All linked requirements 2. Left-click on the model element on the diagram were the rationale or assumption is linked	
Full example	
Synchronise the content to Polarion with the help of Capella2Polarion. The rationale or assumption description will then show up in the work item "Custom Capella Fields".	-

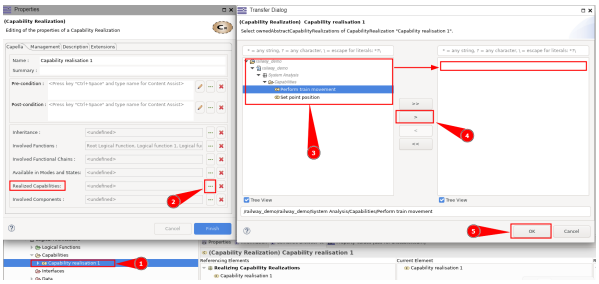
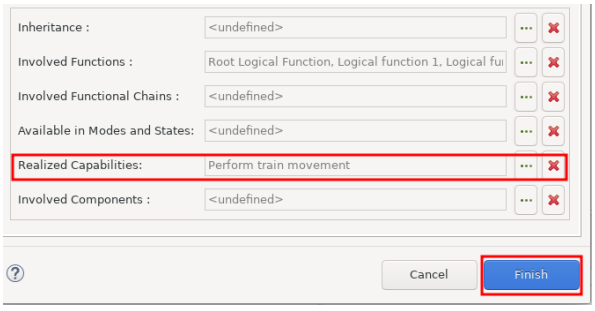
5.3.2 Traceablity

How to establish traceability in between System Analysis and Logical Architecture model elements in Capella

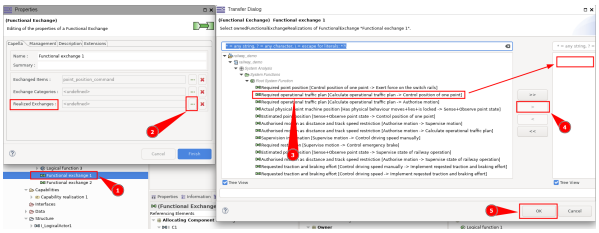
Link logical function to realised system function:

Description	Illustration
To link a logical function to a system function: 1. Double-click left on one logical function in the project explorer or in a diagram 2. Left-click on the "..." button 3. Select the realised system function (In case correct function can not be identified, the conflict should be resolved and consolidated) 4. Move the function to the right side 5. Left-click on the "ok" button Check whether the realisation link is set correctly.	

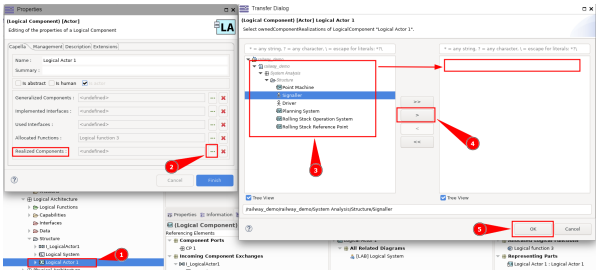
Link logical capability realisation to realised system capability:

Description	Illustration
To link a logical capability realisation to a system capability: 1. Double-click left on one logical capability realisation in the project explorer or in a diagram 2. Left-click on the "..." button 3. Select the realised system capability realisation (In case correct capability can not be identified, the conflict should be resolved and consolidated) 4. Move the capability to the right side 5. Left-click on the "ok" button	
Check whether the realisation link is set correctly.	

Link logical functional exchange to system functional exchange:

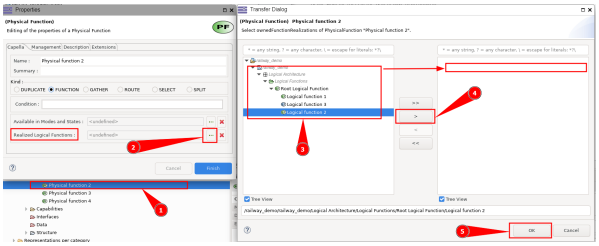
Description	Illustration
To link a logical functional exchange to a system functional exchange: 1. Double-click left on one logical function exchange in the project explorer or in a diagram 2. Left-click on the "..." button 3. Select the realised system functional exchange (In case correct functional exchange can not be identified, the conflict should be resolved and consolidated) 4. Move the functional exchange to the right side 5. Left-click on the "ok" button Check whether the realisation link is set correctly.	

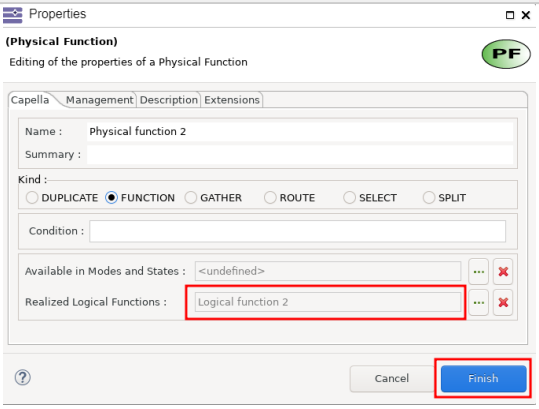
Link logical actor to realised system actor:

Description	Illustration
To link a logical actor to a system actor: 1. Double-click left on one logical function in the project explorer or in a diagram 2. Left-click on the "..." button 3. Select the correct realised system actor (In case correct actor can not be identified, the conflict should be resolved and consolidated) 4. Move the actor to the right side 5. Left-click on the "ok" button Check whether the realisation link is set correctly.	

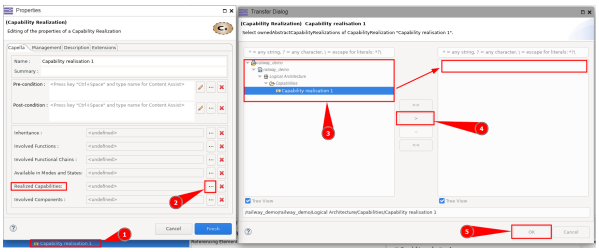
How to establish traceability in between Logical Architecture and Physical Architecture model elements in Capella

Link physical function to realised logical function:

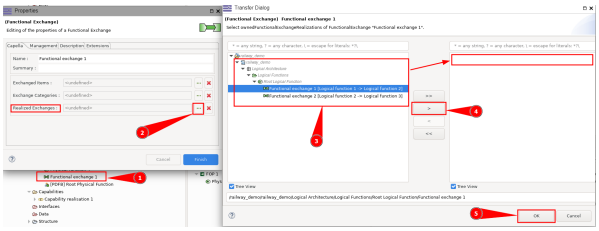
Description	Illustration
To link a physical function to a logical function: 1. Double-click left on one physical function in the project explorer or in a diagram 2. Left-click on the "..." button 3. Select the realised logical function (In case correct function can not be identified, the conflict should be resolved and consolidated) 4. Move the function to the right side 5. Left-click on the "ok" button	
Check whether the realisation link is set correctly.	

Description	Illustration
	

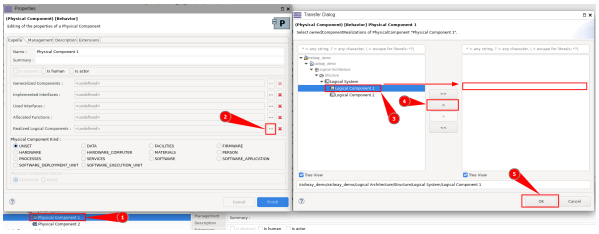
Link physical capability realisation to realised logical capability realisation:

Description	Illustration
To link a physical capability realisation to a logical capability realisation: 1. Double-click left on one physical capability realisation in the project explorer or in a diagram 2. Left-click on the "..." button 3. Select the realised logical capability realisation (In case correct capability can not be identified, the conflict should be resolved and consolidated) 4. Move the capability to the right side 5. Left-click on the "ok" button Check whether the realisation link is set correctly.	

Link physical functional exchange to logical functional exchange:

Description	Illustration
To link a physical functional exchange to a logical functional exchange: 1. Double-click left on one physical function exchange in the project explorer or in a diagram 2. Left-click on the "..." button 3. Select the realised logical functional exchange (In case correct functional exchange can not be identified, the conflict should be resolved and consolidated) 4. Move the functional exchange to the right side 5. Left-click on the "ok" button Check whether the realisation link is set correctly.	

Link physical actor to realised logical actor:

Description	Illustration
To link a physical actor to a logical actor: 1. Double-click left on one physical function in the project explorer or in a diagram 2. Left-click on the "..." button 3. Select the correct realised logical actor (In case correct actor can not be identified, the conflict should be resolved and consolidated) 4. Move the actor to the right side 5. Left-click on the "ok" button Check whether the realisation link is set correctly.	

Link physical component to realised logical component:

Description	Illustration
To link a physical component to a logical component: 1. Double-click left on one physical component in the project explorer or in a diagram 2. Left-click on the "..." button 3. Select the realised logical component (In case correct logical component can not be identified, the conflict should be resolved and consolidated) 4. Move the logical component to the right side 5. Left-click on the "ok" button Check whether the realisation link is set correctly.